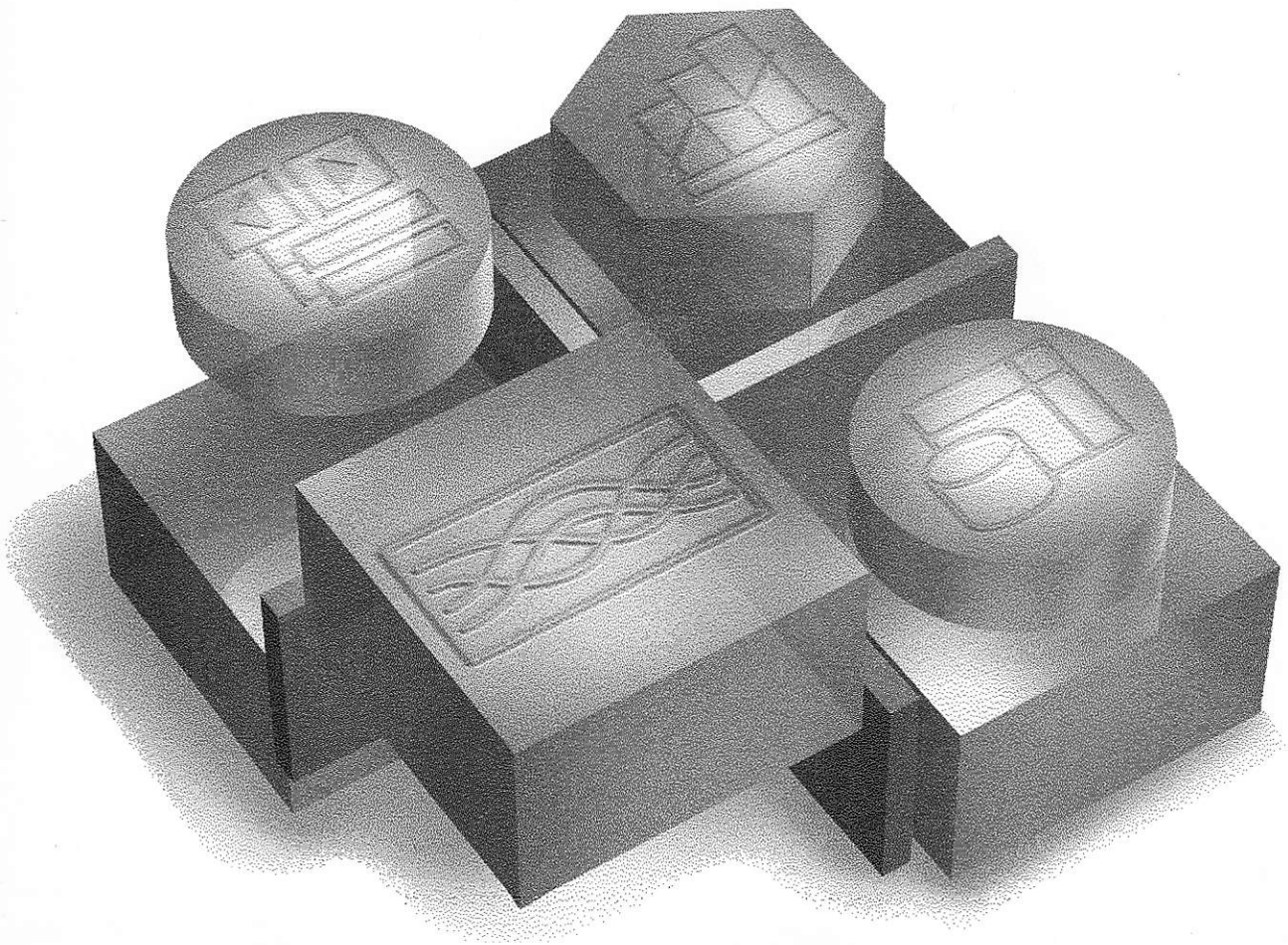
Microsoft®

Visual Basic®
Guide to Data Access Objects
Crystal Reports™ for Visual Basic

Programming System for Windows®



Guide to Data Access Objects

Microsoft® Visual Basic®

Version 5.0
Programming System for Windows® 95 and Windows NT®

Microsoft Corporation

Information in this document is subject to change without notice. Companies, names, and data used in examples herein are fictitious unless otherwise noted. Complying with all applicable copyright laws is the responsibility of the user. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. The furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

©1991–1997 Microsoft Corporation. All rights reserved.

Microsoft® Visual Basic®
Programming System for Windows® 95 and Windows NT®, Version 5.0

Microsoft, ActiveX, FoxPro, MS, MS-DOS, Rushmore, Visual Basic, Windows, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the U.S.A. and other countries.

Other products and company names mentioned herein may be the trademarks of their respective owners.

Document No. DD93015-1296
Printed in Ireland: 03

Contents

Introduction Guide to Data Access Objects vii

Chapter 1 How to Use the Guide to Data Access Objects 1
Introduction to Data Access Objects 3
Data Access in Visual Basic 4

Chapter 2 Creating and Modifying Databases 7
Visual Basic Database Architecture 7
The DAO Object Model 9
Relational Database Design 14
Creating a New Database 19
Attaching Tables from External Databases 26
Modifying a Database 27
Database Maintenance 29

Chapter 3 Working with Records and Fields 35
Using Recordset Objects 35
Moving Through a Recordset 47
Finding a Specific Record 54
Changing Data 63
Microsoft Jet Transactions 70
Extracting Data from a Record 75
Using Field Objects 77

Chapter 4 Incorporating Visual Basic Controls 85
Using the Data Control with Data Access Objects 85
Professional Edition Bound Controls 87
Creating Your Own Bound Controls 87
Using Unbound Controls 89

Chapter 5 Writing SQL Queries 93
What is SQL? 93
The Components of SQL 94
DDL Operations 97
DML Operations 102
Differences Between Jet and ANSI SQL 120

Chapter 6 Database Replication 123
What Is Database Replication? 123
Why Use Database Replication? 123
Replica Set Topology 125
Partial Replication 145
Handling Conflicts and Errors 150
Advanced Design Considerations 157

Chapter 7 Accessing External Data 163
Supported External Data Sources 163
Establishing Access to External Data 167
Working with Desktop Databases 174
Working with Spreadsheets 187
Working with Text Files 196
Working with HTML Files 204
Working with Microsoft Exchange Mailboxes 209
Maintaining Links 213
Performance and Troubleshooting Tips 214

Chapter 8 Creating Multiuser Applications 217
Managing Shared Databases 217
The Microsoft Jet Multiuser Model 220
Locking Shared Data in Recordsets 222
Other Multiuser Design Issues 233

Chapter 9 Developing Client/Server Applications 239
Client/Server Concepts 239
Two Paths to ODBC Data 241
Installing ODBC Drivers 243
Moving Your Data to a Server 244
Accessing ODBC Data With Microsoft Jet 246
Using ODBCDirect 265
Optimizing Performance 285
Other Issues in Client/Server Design 293

Index 301

Introduction

The Guide to Data Access Objects provides detailed information about the Microsoft Jet database engine, the Data control, and the data access objects (DAO) programming interface.

Visual Basic lets you create and manipulate data access objects such as Database, Field, and Index objects, that correspond to the various parts of the physical database you want to access. You use the properties and methods of these objects to perform operations on the database.

DAO lets you write programs that access many different kinds of databases, including *native* databases that use the same format as Microsoft Access, external databases like FoxPro or Paradox, and ODBC client/server databases like Microsoft SQL Server.

CHAPTER 1

How to Use the Guide to Data Access Objects

The *Guide to Data Access Objects* is the primary source of information and instruction on using the Jet database engine and data access objects in your applications. It is intended to be read, rather than used as a reference. A complete reference to all data access topics can be found in Help.

Because the Data control and many of the data-bound controls are included in the Learning Edition of Visual Basic®, their use is covered in the *Programmer's Guide* in Chapter 14, "Accessing Data." The *Guide to Data Access Objects* highlights ways of using these controls with the enhanced features of the Professional Edition.

The *Guide to Data Access Objects* covers the following topics:

Chapter 1, "How to Use the Guide to Data Access Objects," provides a quick overview of the Guide to Data Access Objects and a road map to relevant sections of the documentation for users with specific needs.

Chapter 2, "Creating and Modifying Databases," gives a brief discussion of the Jet engine and the data access objects programming model, and introduces the concepts of relational database design. It also presents the Data Definition Language (DDL) methods for creating new databases, and for modifying existing databases.

Chapter 3, "Working with Records and Fields," discusses the Data Manipulation Language (DML) features of the Jet engine and shows how to use Recordset objects to navigate and manipulate databases, extract information, and perform queries.

Chapter 4, "Incorporating Visual Basic Controls," presents advanced methods for interfacing databases with both bound and unbound controls to create polished user interfaces for Visual Basic database programs. It also shows how to create your own data bound controls in Visual Basic.

Chapter 5, "Writing SQL Queries," provides a brief tutorial on the SQL query language. Example queries are presented along with a discussion of query optimization and a summary of the SQL syntax used in the Jet database engine.

Chapter 6, "Database Replication," discusses the tools and strategies built in to DAO and the Jet engine for creating, maintaining, and synchronizing database replicas.

Chapter 7, "Accessing External Data," discusses the details of accessing data in external databases like Microsoft® Access, Btrieve, dBASE, FoxPro®, and Paradox, as well as SQL Server and other ODBC databases. Working code examples and initialization (.ini) file listings are provided for each supported database format.

Chapter 8, "Developing Multiuser Applications," discusses special considerations for creating and working with Jet databases that will be shared by multiple users.

Chapter 9, "Developing Client/Server Applications," presents the ODBCDirect mode of Data Access Objects, and discusses many of the techniques used in creating true client/server applications.

For the Novice Programmer

Programmers with little database experience should begin by reading Chapter 14, "Accessing Data" in the *Programmer's Guide*, and working through the examples presented there. The Data control is a simplified interface to the Jet engine that does not require the use of data access objects for simple database programming.

When you are ready to tackle the enhanced capabilities of the DAO programming model, you would do well to read this book from cover to cover for the best possible overview. At the very least, you should read the first four chapters, and follow the example code carefully. Following this, you can refer to specific topics in the remainder of the book as the need arises.

An additional source of information are the *Language Reference* and *Control Reference* in Books Online. This is especially useful when you have a general idea how to accomplish some task, but are "stuck" for the exact details. Books Online features a complete reference to all data access objects, their properties, methods, syntax, and options.

For the Experienced Database Programmer

If you are an experienced Visual Basic programmer and are familiar with database concepts and the previous version of the Jet database engine, you need not read through all the introductory material in "Creating and Modifying Databases," although the presentation of the data access object model may give you a useful overview. The real "meat" of database programming is covered in "Working with Fields and Records," "Incorporating Visual basic Controls, and "Writing SQL Queries." For building multiuser and client/server applications, "Developing Multiuser Applications," and "Developing Client/Server Applications" are essential.

Sample Applications

The sample applications provided with Visual Basic provide an excellent learning tool, as well as a ready source of well written code that you can cut and paste into your application and modify as necessary. The most useful sample applications for creating data access applications are:

- Biblio.vbp
- Visdata.vbp

The Biblio.vbp example is used extensively in this manual. The Visdata.vbp example is a handy reference because it contains examples of every supported data access object and method. Visdata is also provided as an add-in utility on the Add-Ins menu, and provides a useful general-purpose front end to your databases.

Introduction to Data Access Objects

All applications use structured information of one kind or another, whether it is accounting data, scientific measurements, employee information, or a list of recipes. Data access in Microsoft Visual Basic gives you the tools to create and use structured database systems to manage your application's data.

These tools include the Microsoft Jet database engine, the Data control, and the data access objects (DAO) programming interface.

Using data access objects, you can create databases and build full-featured applications that access existing databases in many popular formats, including Microsoft Access, Btrieve, dBASE, Microsoft FoxPro, and Paradox, as well as Open Database Connectivity (ODBC) client/server databases like Microsoft SQL Server.

The ability to create and access structured database systems gives you many programming advantages:

- It lets you write programs that use existing databases.
- It allows your application to share data with other programs.
- It simplifies your programming, since you do not need to handle low-level file accessing and searching.

Data Access in Visual Basic

Visual Basic provides two methods of interfacing with the Jet database engine: the Data control and data access objects. The Data control is introduced in the *Programmer's Guide* in Chapter 14, "Accessing Data." While the Data control gives you limited ability to access existing databases without programming, the DAO model is a complete programming interface that gives you total control of the database. These two methods are not mutually exclusive; in fact, there are many situations where you will want to use both of them together.

The DAO model is a collection of object classes that model the structure of a relational database system. They provide properties and methods that allow you to accomplish all of the operations necessary to manage such a system, including facilities for creating databases, defining tables, fields and indexes, establishing relations between tables, navigating and querying the database, and so on.

The Jet database engine translates these operations on data access objects into physical operations on the database files themselves, handling all the mechanics of interfacing with the different supported databases.

Database programming in Visual Basic consists of creating data access objects, such as Database, TableDef, Field, and Index objects, that correspond to the various parts of the physical database you want to access. You use the properties and methods of these objects to perform operations on the database. You can display the results of these operations and accept input from the user on Visual Basic forms, using both bound and unbound controls.

This approach simplifies the code you need to write and insulates you from the underlying structure and mechanics of retrieving and updating data. It gives you great flexibility, because you can use the same objects, properties, and methods to work with a wide variety of supported database formats. Also, if you change from one database format to another (for example, porting a local Microsoft Access database to a SQL Server database on a network), you'll need to make few changes in your code to accommodate the change. You can even create applications that join tables from two or more different databases in a single query or report.

There are three categories of databases that Visual Basic recognizes through DAO and the Jet engine:

- Visual Basic Databases

  Also called *native* databases, these database files use the same format as Microsoft Access. These databases are created and manipulated directly by the Jet engine and provide maximum flexibility and speed.

- External Databases

  These are Indexed Sequential Access Method (ISAM) databases in several popular formats, including Btrieve, dBASE III, dBASE IV, Microsoft FoxPro versions 2.0 and 2.5, and Paradox versions 3.x and 4.0. You can create or manipulate all of these database formats in Visual Basic. You can also access text file databases and Microsoft Excel or Lotus 1-2-3 worksheets.

- ODBC Databases

  These include client-server databases that conform to the ODBC standard, such as Microsoft SQL Server. To create true client-server applications in Visual Basic, you can use ODBCDirect to pass commands directly to the external server for processing.

Note There are other methods of data access supported by Visual Basic, which do not use the Jet database engine. The ODBCDirect mode of Data Access Objects allows you to access ODBC data directly, using the same object model and syntax but bypassing the Jet engine. In addition, the Remote Data Objects (RDO) library and Remote Data control (RDC) are included in the Enterprise Edition, and allow direct access to ODBC data. Finally, there are the ODBC Libraries, which allow you to call the ODBC applications programming interface (API) directly, and the Visual Basic SQL Libraries (VBSQL), which provides a direct link to the Microsoft SQL Server API. The toolkits for these libraries are available as separate Microsoft products.

For More Information For additional information on the ODBCDirect mode of DAO, see Chapter 8, "Developing Client/Server Applications." For more information about Remote Data Objects, the ODBC API, and VBSQL, see Part 3, "Data Access Options," in the *Guide to Building Client/Server Applications.*

What's New in DAO for Version 5.0

Visual Basic 5.0 incorporates version 3.5 of the Jet database engine for 32-bit applications. Because version 5.0 is a 32-bit only programming language, there is no 16-bit Jet engine or DAO library. However, you will find that DAO version 3.5 is fully backward compatible, making it easy to upgrade your 16-bit applications.

Aside from significant performance improvements, there are two important new features. These are:

- ODBCDirect, an alternate mode of DAO that allows you to access ODBC data sources directly, bypassing the Jet engine and taking full advantage of the remote data source's processing capabilities. ODBCDirect is discussed in Chapter 9, "Developing Client/Server Applications."
- The ability to create your own data bound ActiveX™ controls in Visual Basic, using the DataBindings collection. This feature is presented, along with examples, in Chapter 4, "Incorporating Visual Basic Controls."
- The ability to bind multiple data sources to a single control. This provides a convenient mechanism for storing persistent information about your control's behavior that the user can alter at run time. This feature is also discussed in Chapter 4, "Incorporating Visual Basic Controls."

Differences in Learning, Professional and Enterprise Editions

The Learning Edition of Visual Basic incorporates a limited subset of the Jet engine's data access capabilities in the Data control and special data-bound controls. These features allow you to access existing databases, but do not provide for creating new databases or data access objects.

The Professional Edition includes all of the Data control features, and in addition provides the full data access objects programming interface. Professional Edition users can combine data access objects with the Data control to gain all of the advantages of simplified data access while maintaining the flexibility of complete programmatic control.

The Enterprise Edition adds a number of powerful Client/Server tools to Visual Basic's data access capabilities, including the Remote Data control (RDC) specifically designed to access remote ODBC client/server databases. For more information on the Enterprise Edition, see *Building Client/Server Applications* with Visual Basic.

CHAPTER 2

Creating and Modifying Databases

This chapter introduces the Microsoft Jet database engine and its programming model, Data Access Objects (DAO). It presents the basic concepts of relational database design and shows how to create, maintain, and modify databases using the data definition language (DDL) methods of DAO. It explains how to use DAO to define database tables, fields, and indexes, and how to create relations between tables. Also included are techniques for modifying database structures and maintaining existing databases.

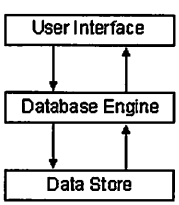
The Microsoft Jet Database Engine

Visual Basic provides data access capability based on the Microsoft Jet database engine, the same database engine that powers Microsoft Access. The Jet engine handles the mechanics of storing, retrieving, and updating data, and provides you with the powerful DAO object-oriented programming interface.

Visual Basic version 5.0 comes with version 3.5 of the Microsoft Jet database engine, a 32-bit only engine. Because it is backward-compatible with earlier versions of the Jet database engine, your 16-bit applications written in earlier versions of Visual Basic can easily be upgraded to take advantage of the additional features and performance improvements in version 5.0.

Visual Basic Database Architecture

A Visual Basic database application has three parts, as shown below.



The database engine lies between your program and the physical database files. This gives you independence from the particular database you are accessing. Whether it is a native Visual Basic database or any of the other supported database formats, you use the same data access objects and programming techniques.

User Interface and Application Code

The user interface is what the user sees and interacts with. It contains forms that display the data and enable the user to view or update it. Driving these forms is your application's Visual Basic code, including data access objects and methods to request database services like adding or deleting records, or performing queries.

As the illustration above shows, these requests for services are not directed to the physical database file, but to the Jet database engine, which performs the requested operations on the data store and returns the desired results to the application.

The Database Engine

The Jet Database engine is contained in a set of dynamic link library (DLL) files that are linked to your Visual Basic program at run time. It translates the application's requests into physical operations on the .mdb file or other data store. It is what actually reads, writes, and modifies the database, and handles all the housekeeping chores like indexing, locking, security, and referential integrity. It also contains a query processor that accepts your Structured Query Language (SQL) queries and acts on them to carry out the desired database operations, and a results processor to manage the results returned by queries.

The Data Store

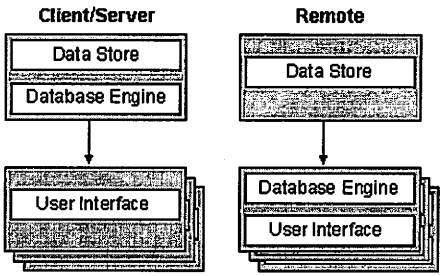
The data store is the file or files containing the database tables themselves. For native Visual Basic or Microsoft Access databases, this is an .mdb file. For different databases, it might be a directory containing files with .dbf or other extensions. Or, your application might access data stored in several different database files and formats. In any case, the data store is essentially passive. It contains the data, but doesn't *do* anything to or with it. That is the job of the database engine.

Local and Remote Databases

The three components can be divided in many ways. All three parts can be present on one computer in a single-user application. Or, they can be located on different computers connected over a network. For example, the data store might reside on a central file server, and the user interface (the "application program") on several workstations, giving many users access to the same data.

Remote vs. Client/Server

An important distinction in database applications where the data is stored on a different machine from the application program, is between a *remote* database and a *client/server* database. The difference is illustrated below.



In a client/server system, the database engine is located on a central server, along with the data store. The engine can service many "client" applications simultaneously, manipulating the data store and returning the requested records to each local application. In a remote system, the database engine is on the same computer as the user application, with only the data store residing on a remote machine.

The Jet database engine is not a client/server engine. It is a local database engine residing in DLL files that are dynamically linked to your Visual Basic program at run time. If there are many copies of your program on different workstations, each will have its own local copy of the Jet database engine, even though they may all be accessing a single remote database. This has important performance considerations, which will be discussed at greater length in Chapter 7, "Accessing External Data."

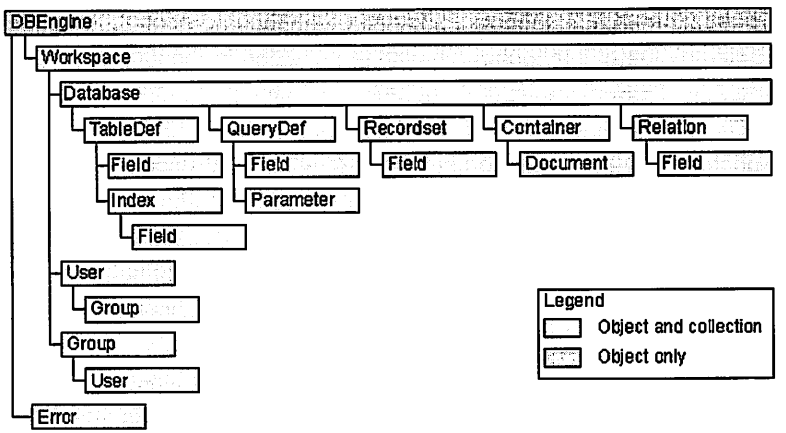
Note It is possible to create client/server applications in Visual Basic by connecting to an Open Database Connectivity (ODBC) data source, such as Microsoft SQL Server, and passing your queries directly to the server database engine or using remote stored procedures. For more information on this topic, refer to Chapter 7, "Accessing External Data."

The DAO Object Model

The data access object model is the Jet database engine's object-oriented interface. It is a hierarchy of classes that correspond to a logical view of a relational database system, such as the database itself, the tables defined in it and their fields, indexes, and so on. These classes are used to create data access objects that refer to the particular database you want to manipulate.

For consistency and ease of use, these classes and the objects created from them behave like other classes and objects in the Visual Basic environment. They are addressed using the familiar Visual Basic syntax and manipulated through their properties and methods, as described in the *Programmer's Guide*, Chapter 9, "Programming with Objects."

The data access object classes are organized as a hierarchy, in which most classes belong to a collection class, which in turn belongs to another class above it in the hierarchy. The entire DAO hierarchy is illustrated here.



Classes, Objects, and Collections

The notion of an object hierarchy simply means that objects can contain other objects, which can in turn contain other objects. This "containment" is accomplished through a special kind of object called a *collection*. The sole purpose of a collection object is to contain other objects. The objects contained by a given collection are all of the same kind.

Note The elements in the DAO hierarchy are actually *classes*, not *objects*. They are the "blueprints" for the objects that you create to build your database application. A class is similar to a data type, in that it describes what "kind" of object you are referring to. For example, when you declare

```
Dim MyWs As Workspace
```

you are stating that `MyWs` is a variable that stands for an object of the Workspace class.

Since most of the discussion in this chapter concerns the behavior of the objects you create, the term "object" is used in preference to "class." Just remember that the term "Database object," for example, means "an object of the Database class."

As the illustration shows, most of the data access objects are represented as both "object *and* collection." At the top of the hierarchy is the Microsoft Jet database engine itself, the DBEngine object. It is the only data access object that is not contained in anything. It owns a collection called Workspaces (the name of a collection is always the plural of the objects it contains), which contains one or more Workspace objects. Each Workspace object has a Databases collection, which contains one or more Database objects. Each Database object has a TableDefs collection, which contains one or more TableDef objects, and so on.

Collection Syntax

For a complete discussion of object and collection syntax, see the *Programmer's Guide*, Chapter 9, "Programming with Objects."

The member objects of a collection can be accessed through a zero-based index. For example, the first TableDef in a Database called MyDatabase is referred to as MyDatabase.TableDefs(0). The second TableDef in the same Database is MyDatabase.TableDefs(1), and so on.

Objects in the hierarchy are identified by the full "path" through the nested collections to which they belong, using the . (dot) navigation operator. Thus,

```
DBEngine.Workspaces(0).Databases(0).TableDefs(0).Fields("Customer")
```

refers to the field named Customer in the first TableDef in the TableDefs collection of the first Database in the Databases collection of the first Workspace in the Workspaces collection of the DBEngine. Some collections also permit other indexes. As shown in the preceding example, the Fields collection also enables you to index using a string, corresponding to the Field object's Name property.

When you refer to a collection member explicitly (as opposed to using an index to the collection), use the ! (bang) operator instead of the dot. Thus, the following expressions are equivalent:

```
MyTableDef.Fields("Customer")
MyTableDef.Fields!Customer
```

Default Collections

Most data access objects have *default* collections. These enable you to simplify your code, because you do not have to explicitly provide the collection name when referring to an element from the default collection. For example, the default collection of the Recordset object is the Fields collection. To get the value of a Field named Customer, you can simply write:

```
Cust = MyRecordset!Customer    ' Gets value of
                               ' "Customer" field.
```

which is shorter than

```
Cust = MyRecordset.Fields!Customer
```

–or–

```
Cust$ = Myrecordset.Fields("Customer")
```

Default collections simplify your code when making nested object references, because the default collections are the ones that implement the main hierarchy of data access objects.

Properties and Methods

One of the main ideas of object oriented programming is that all of the data and procedures related to a particular object are kept together with the object itself. In Visual Basic, an object's data (settings or attributes) are called *properties*, while the various procedures that can operate on the object are called its *methods*.

Programming with data access objects consists of creating object variables, and manipulating them by invoking their methods and setting their properties.

For example, consider the following code fragment:

```
Dim MyDB As Database, MyWS As Workspace, _
MyRS As Recordset

Set MyWS = DBEngine.Workspaces(0)
Set MyDB = MyWS.OpenDatabase("Accounts.mdb")
Set MyRS = MyDB.OpenRecordSet("Clients")

MyRS.Index = "ClientID"
```

1. The OpenDatabase method of the MyWS Workspace object is used to open the Accounts.mdb database, and assign it to the MyDB Database object variable.
2. Next, MyDB's OpenRecordset method is used to create a recordset (a group of records representing a database table or the results of a query) based on the Clients table, and assign it to the MyRS Recordset variable.
3. Finally, the Index property of the MyRS Recordset object is set to the ClientID Index.

Other methods of the Recordset object could be used to navigate through the records, add a new record to the table, edit a record or delete a record. Other properties could be examined or set to filter the Recordset, determine when it was last updated, and change other attributes of the Recordset.

Writing a value into a property may cause the object to take a specific action. In the preceding example, setting the Recordset object's Index property causes the logical sort order of the Recordset to instantly change to match the specified index.

The user can define new properties on most data access objects. This is a powerful programming tool that enables you to customize objects and save the changes permanently with the object in the database.

Divisions of Database Programming

Now that you have been introduced to the basic concepts of database programming with data access objects and the Jet engine, the next step is to see how the various pieces fit together to create database applications. There are two broad areas into which all database programming tasks can be categorized. The common terms for these areas are:

- Data Definition Language (DDL)
- Data Manipulation Language (DML)

This doesn't mean that there are two separate languages involved. It is simply a way of grouping the language elements—in this case, data access objects, properties, and methods—that are used to define the structure of a database, and those that are used to manipulate it.

Data Definition

DDL consists of those properties and methods that are used to define and create the database itself, including its tables, fields, relations, and so on. In traditional database terminology, this set of definitions constitutes the database's *schema*.

Defining the database is typically a one-time operation, because the Jet engine stores the data access objects that define the database schema in the database itself. Once the database has been created, there is no need to specify its structure in order to access it. Simply opening the database brings all of its objects, including those that define its structure, under the management of the DAO interface.

Data Manipulation

DML consists of the properties and methods that you use to write applications that access and manipulate existing databases. This includes facilities for querying the database, navigating through its tables, performing updates, and adding or deleting records.

If you are working exclusively with databases that have already been created by another application, it is possible to create applications entirely within the DML functionality. Understanding the DDL methods, however, will add to your knowledge of database structure and make you a more flexible database programmer.

Relational Database Design

There are many ways of organizing databases. Among the most popular are indexed sequential access method (ISAM) file systems, network-model databases, hierarchical databases, and relational databases. These types of databases differ not only in the way they physically manage the storage and retrieval of data, but also in the conceptual models they present to the user and programmer.

In recent years, the relational model has generally become the de facto standard for database design. This is due both to the power of the relational model itself, and because it provides a standard interface called Structured Query Language (SQL) that allows many different database tools and products to work together in a consistent and understandable way. The Microsoft Jet database engine is a relational database engine.

The Structure of Relational Databases

The relational database model presents the data as a collection of tables. Instead of modeling the relationships in the data according to the way that it is physically stored, the structure is defined by establishing *relations* between simple tables.

Tables, Fields, and Records

Regardless of how it's physically stored in the database file, a table can be viewed as a set of rows and columns, similar to the rows and columns of a spreadsheet. In a relational database, the rows are called *records* and the columns are called *fields*. For example, a table of customers might look like:

Customer Table

CustID	LastName	FirstName	Address	City	ST	Zip
1723	Doe	John	1234 Fifth Avenue	New York	NY	10043
3391	Smith	Mary	9876 Myrtle Lane	Boston	MA	60781
3765	Blaselflatz	Mortimer	2294 River Road	Peoria	IL	70115

Each row in the table is a record that contains all of the information about a particular customer, and each record contains the same types and number of fields: CustID, LastName, FirstName, and so on.

Keys

A *key* is a field or fields in the table that is indexed for fast retrieval. A key can be unique or non-unique, depending on whether duplicates are allowed. A unique key can be designated as the *primary key*, designating it as the unique identifier for each row of the table. In the preceding example, for instance, the customer identification number (CustID) is the table's primary key, because the CustID uniquely identifies one and only one customer.

Relations

A database can be composed of more than one table, and the tables can be related to one another in various ways. For example, the customer database might also have a table listing all of the orders placed by a particular customer. Rather than repeating all of the customer information for each entry in the orders table, it could contain a single field that referred to the customer who placed the order, as shown in the following table.

Order Table

Order	CustID	Date	Item	Amount
14764	3391	2/23/96	27	$22.95
14932	3391	3/17/96	46	$9.57
15168	8765	2/15/96	27	$22.95

In this table, the CustID field refers to the CustID field in the customer table, relating the order to the customer who placed it. We can see that customer 3391 (Mary Smith) ordered Item 27 on 2/23/94 and Item 46 on 3/17/94. The key that establishes the relation is called a *foreign key*, because it relates to the primary key of a "foreign" table (the Customers table).

One-to-Many and Many-to-Many

The type of relation shown in the preceding table is called a *one-to-many* relation, because one customer can place many orders, but a particular order is placed by one and only one customer. It is also possible to establish relations that are *many-to-many*. For example, there may be an inventory table that lists all of the items available for sale, as follows.

Inventory Table

Item	Description	Supplier	Cost	Inventory
27	Straw Hat	Garden Supply Co.	$14.00	50
46	Garden Gloves	Garden Supply Co.	$4.50	75
102	Hanging Planter	Floral Industries	$6.00	137

Here we can see that there is a many-to-many relationship between customers and inventory items. That is, one customer can order many inventory items, and one inventory item can be ordered by many different customers. A many-to-many relationship is defined by creating two separate one-to-many relationships, with the common "many" table containing foreign keys to both of the other tables. In this example, the Orders table is related to both the Inventory table (through Item) and the Customer table (though CustID). By looking at all three tables, we can see that Mary Smith (CustID 3391) ordered both a straw hat (Item 27) and garden gloves (Item 46), *and* that straw hats (Item 27) were ordered by both Mary Smith (CustID 3391) and Mortimer Blaselflatz (CustID 8765). This relationship is clearer if we create a "junction" table by *joining* the relevant fields from the Customer and Inventory tables to the Order field from the Orders table.

Junction Table: Orders By Customer, Item

Order#	CustID#	LastName	FirstName	Item#	Description
14764	33391	Smith	Mary	27	Straw Hat
14932	33391	Smith	Mary	46	Garden Gloves
15168	8765	Blaselflatz	Mortimer	27	Straw Hat

For more information about joining tables and other relational operations, see Chapter 6, "Writing SQL Queries."

Normalization

The task of the database designer is to structure the data in a way that eliminates unnecessary duplication and provides a rapid search path to all necessary information. The process of dividing the information into separate tables that meet these goals is called *normalization.*

Normalization can be a complex process with many specific rules and different levels of *normal form.* A complete discussion of the process is beyond the scope of this manual. However, normalizing most simple databases can be accomplished by following a simple rule of thumb: tables that contain repeated information should be divided into separate tables to eliminate the duplication.

For example, a student database that matches students to courses contains the information shown in the following table.

Student	Course	Course description	Instructor
1	4	Introduction to Physiology	Dawson
2	3	Applied Basketweaving	Carruth
3	1	Physics for Short-Order Cooks	Adams
4	4	Introduction to Physiology	Dawson

If there are 1,000 students taking a dozen courses, each course description and instructor will appear 100 or more times — once for each student who takes that course. To avoid this inefficiency, the table should be normalized by dividing it into two separate tables, one for students and one for courses, as follows.

Student	Course
1	4
2	3
3	1
4	4

Course	Course description	Instructor
1	Physics for Short-Order Cooks	Adams
2	Counterculture Sociology	Beckley
3	Applied Basketweaving	Carruth
4	Introduction to Physiology	Dawson

Because the tables are normalized, changing the Course Description or Instructor for a particular course can now be accomplished by changing a single record.

Designing a Database

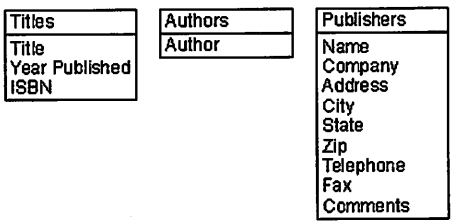
For a brief example, you can look at the design of the Biblio.mdb sample database included with Visual Basic version 5.0. This is a bibliography of books about database programming.

The information you will want to track about each book includes:

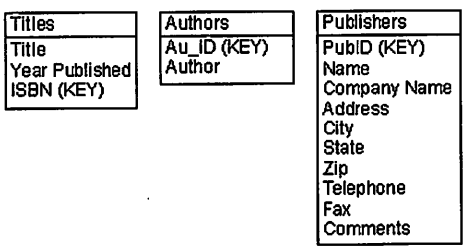
- Author
- Title
- Year Published
- ISBN (International Scientific Book Number)
- Publisher (Short bibliographic name)
- Company Name (Publisher's full business name)
- Address
- City
- State
- Zip
- Telephone
- Fax
- Publisher's comments

Of course, you could simply create a single table with a field for each of the data items listed. Looking closely at the data, however, it's obvious that such a table would contain many redundancies. For example, many authors have written more than one book, and many publishers have published more than one book on database programming. If you put all fields into a single table, there would be many duplicate entries in the Author field and in all of the fields that relate to the Publisher.

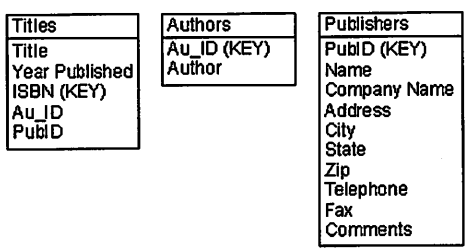
Accordingly, you can break the table into three groups: Titles, Authors, and Publishers, as shown in the following illustration.



Now you need a means of relating the tables to one another. First, assign primary keys to each table. The Titles table already has the ISBN as a natural unique key, and you add Au_ID and PubID as unique keys to the Authors and Publishers tables, respectively.



Finally, you add Au_ID and PubID to the Titles table as foreign keys into the Authors and Publishers tables, respectively.



This schema creates one-to-many relationships between the Authors and Titles tables (an author may write many titles) and between the Publishers and Titles tables (a publisher may publish many titles). It also creates a many-to-many relationship between Authors and Publishers (an author's titles may be published by many publishers, and a publisher's titles many be written by many authors).

Creating a New Database

After the database is designed, the next step is to create it.

DAO isn't the only method of creating a new database. Other possible approaches include:

- The Data Manager application included with Visual Basic. With the Data Manager, you can create a Jet database without programming. Once the database is created, you can modify and manipulate it using the methods discussed later in this chapter and in Chapter 4, "Accessing and Navigating Databases." For information on using the Data Manager application, see Help in the Data Manager.
- Microsoft Access. Because it uses the same database engine and format as Visual Basic, databases you create with Microsoft Access can be used just as if you had created them directly in Visual Basic. (The exception to this rule is that Jet version 3.0 databases created by Access can only be used with Visual Basic applications that also use the Jet version 3.0 engine.) For more information about the two versions, see the discussion that follows under "16-Bit or 32-Bit."
- An external database application. Products such as FoxPro, dBASE, or ODBC client/server applications can create a new external database that Visual Basic can access through ISAM or ODBC drivers.
- Create a new directory. For ISAM databases, simply creating a new directory in Windows® using File Manager, or by using the MKDIR command in MS-DOS®, essentially "creates" a new database. You will still need to use the external database application or data access objects to add tables, records, and fields. See "Modifying the Database" later in this chapter for more information.

Despite the many alternatives, this chapter focuses on using the self-contained tools in Visual Basic to create a new database. DAO gives you the greatest depth of programming control over a database. It is also the method of choice if you want your application itself to be able to create a new database at run time.

Note You can also create new databases from within your Visual Basic code using SQL statements that implement DDL. For more information about SQL statements, see Chapter 5, "Writing SQL Queries."

Steps to Create a New Database

The procedure for creating a new Jet database is simply a process of creating and defining data access objects that correspond to the tables, fields, indexes, and relations of your database design. In this overview, the process will be presented step-by-step, followed by a complete code example for creating the Biblio.mdb database. For a complete discussion of the syntax and options for each method discussed, see the *Language Reference* in Books Online.

The first step in creating a new database is to create the Database object itself, and to define its structure by adding TableDef and Field objects that correspond to your design.

▶ To create a new database

1 Use the Dim statement to create new object variables for each object in your database. In addition to the DBEngine and default Workspace objects that make up the working environment, you will need:

- One Database object
- One TableDef object for each table
- One Field object for each field in each table
- One Index object for each index in each table

For example, to create object variables for the Biblio.mdb database, you could use the following code:

```
Dim MyDB As Database, MyWs As Workspace
Dim AuTd As TableDef, TitTd As TableDef, _
PubTd As tableDef
Dim AuFlds(2) As Field, TitFlds(5) _
As Field, PubFlds(10) As Field
Dim AuIdx As Index, TitIdx(3) As Index, _
PubIdx As Index
```

2 Use the CreateDatabase method of the Workspace object to create the new database. In this example, the method uses the minimum two arguments: one to specify the database name, and one to specify the locale:

```
Set MyWs = DBEngine.Workspaces(0)
Set MyDb = MyWs.CreateDatabase("C:\VB\Biblio.mdb", _
dbLangGeneral, dbVersion30)
```

Note that the constant dbVersion30 specifies a Jet version 3.0 database. If you use the dbVersion30 constant to create a version 3.0 database, only 32-bit applications using the Jet version 3.0 engine or higher will be able to access it.

3 Use the CreateTableDef method of the Database object to create new TableDef objects for each table in the database, as follows:

```
Set TitTd = MyDB.CreateTableDef("Titles")
Set AuTd = MyDB.CreateTableDef("Authors")
Set PubTd = MyDB.CreateTableDef("Publishers")
```

4 Use the CreateField method of the TableDef object to create new Field objects for each field in the table, and to set properties of each field to define the field's size, data type, and other needed attributes. For example, the following code creates the Authors table in the Biblio.mdb database:

```
Set AuFlds(0) = AuTd.CreateField("Au_ID", dbLong)
' Make it a counter field.
AuFlds(0).Attributes = dbAutoIncrField
Set AuFlds(1) = MyTd.CreateField("Author", dbText)
AuFlds(1).Size = 50
```

5 Use the Append method to add each field to its table and each table to the database, as shown in the following code:

```
AuTd.Fields.Append AuFlds(0)
AuTd.Fields.Append AuFlds(1)
MyDB.TableDefs.Append AuTd
```

Note In this example, a different variable is used for each data access object, for clarity. In practice, you can reuse variables to simplify your code. For example, you can use the Dim statement to declare a single TableDef variable, use CreateTableDef to assign a new TableDef object to it, and append it to the Database. Once it is appended, you can use the same TableDef variable to create and append each new TableDef.

Adding Indexes and Relations

You add indexes to the TableDef by creating them with the CreateIndex method. In order to specify which fields are to be indexed, you create new Field objects with the CreateField method of the Index object.

▶ To add an index to a database table

1 Create indexes for each table using the CreateIndex method of the TableDef object, and set their properties:

```
Set AuIdx = MyTd.CreateIndex("AuthorID")
AuIdx.Primary = True
AuIdx.Unique = True
```

2 Create fields for each index object using the CreateField method of the Index object:

```
Set NewFld = AuIdx.CreateField("Au_ID")
```

3 Append the field to the Index and the Index to the TableDef object:

```
AuIdx.Fields.Append NewFld
MyTd.Indexes.Append AuIdx
```

Note Fields created with the CreateField method of the Index object aren't added to the TableDef. Instead, they are added to the Index object, and given the same Name property as the TableDef field that they are to index. You don't specify Type and Size properties on fields of the Index object.

Relations and Referential Integrity

Creating TableDefs that implement primary and foreign keys, as described previously, allows you to relate records in one table to the corresponding records in another table, according to the common primary/foreign key values. For the relation to be useful, however, it is vital to maintain *referential integrity* when adding or deleting records. Referential integrity simply means that the foreign key(s) in any referencing table must always refer to a valid record in the referenced table.

For example, in the Biblio.mdb database, the PubID field of the Titles table is a foreign key that references the PubID key of the Publishers table. If there is a record in the Titles table with a PubID entry of, for example, 25, there must actually *be* a record in the Publishers table with a PubID of 25. If there isn't, there is no way to determine which publisher published this particular title and the database is effectively corrupted. This type of corruption is called a referential integrity violation.

The Jet database engine provides the Relation object to enforce referential integrity and to prevent such violations.

▶ To add a relation to a database

1 Create a Relation object using the CreateRelation method of the Database object and set its Table and ForeignTable properties:

```
Dim Au_Tit As Relation
Set Au_Tit = MyDb.CreateRelation("Authors_Titles")
Au_Tit.Table = "Authors"
Au_Tit.ForeignTable = "Titles"
```

2 Create a field to define the common primary/foreign key field in the relation, using the CreateField method of the Relation object:

```
Dim TempField As Field
Set TempField = Au_Tit.CreateField("Au_ID")
TempField.ForeignName = "Au_ID"
```

3 Use the Append method to add the Field object to its Relation and the Relation to the Database.

```
Au_Tit.Fields.Append TempField
MyDb.Relations.Append Au_Tit
```

Setting Additional Properties

The Create methods discussed in these topics allow you to set additional properties and options for the created objects. You can find a complete reference to all objects, properties, and methods, along with code examples, in the *Language Reference* in Books Online.

If you omit one or more of the optional arguments when you use one of the Create methods, you can use an assignment statement to set or reset the corresponding property before you append the new object to a collection. For example,

```
Set MyFld = MyTableDef.CreateField("ClientName", _
dbText, 30)
```

is equivalent to

```
Set MyFld = MyTableDef.CreateField()
MyFld.Name = "ClientName"
MyFld.Type = dbText
MyFld.Size = 30
```

Code Example: Creating Biblio.mdb

As an example of creating a new database in Visual Basic code, the following procedure was used to create the Biblio.mdb database.

```
Sub MakeBiblio ()
Dim MyDB As Database, MyWs As Workspace, _
lxFlds(2) As Field
Dim i As Integer
Dim AuTd As TableDef, TitTd As TableDef, _
PubTd As TableDef
Dim AuFlds(2) As Field, TitFlds(5) As Field, _
PubFlds(10) As Field
Dim AuIdx As Index, TitIdx(3) As Index, PubIdx As Index

Set MyWs = DBEngine.Workspaces(0)
Set MyDB = MyWs.CreateDatabase("C:\VB\Biblio.mdb", _
dbLangGeneral, dbVersion25)
' Create new TableDef for Authors table.
Set AuTd = MyDB.CreateTableDef("Authors")
' Add fields to MyTableDef.
Set AuFlds(0) = MyTd.CreateField("Au_ID", dbLong)
' Make it a counter field.
AuFlds(0).Attributes = dbAutoincrfield
Set AuFlds(1) = MyTd.CreateField("Author", dbText)
AuFlds(1).Size = 50
AuTd.Fields.Append AuFld(0)
AuTd.Fields.Append AuFld(1)
' Now add an Index.
```

```
Set AuIdx = AuTd.CreateIndex("Au_ID")
AuIdx.Primary = True
AuIdx.Unique = True
Set IxFlds(0) = AuIdx.CreateField("Au_ID")
' Append Field to Fields collection of Index object.
AuIdx.Fields.Append IxFlds(0)
' Append Index to Indexes collection.
AuTd.Indexes.Append AuIdx
' Append TableDef to TableDefs collection.
MyDB.TableDefs.Append AuTd

' Create new TableDef for Titles table
Set TitTd = MyDB.CreateTableDef("Titles")
' Create fields for Titles Table
Set TitFlds(0) = TitTd.CreateField("Title", dbText)
TitFlds(0).Size = 100
Set TitFlds(1) = TitTd.CreateField("Year Published", _
dbInteger)
Set TitFlds(2) = TitTd.CreateField("ISBN", dbText)
TitFlds(2).Size = 20
Set TitFlds(3) = TitTd.CreateField("Au_ID", dbLong)
Set TitFlds(4) = TitTd.CreateField("PubID", dbLong)
' Append fields to Titles TableDef.
For i = 0 to 4
   TitTd.Fields.Append TitFld(i)
Next i
' Now add indexes.
For i = 0 to 2
   Set TitIdx(i) = TitTd.CreateIndex()
Next i
TitIdx(0).Name = "ISBN"
TitIdx(0).Primary = True
TitIdx(0).Unique = True
Set IxFld(0) = TitIdx(0). CreateField("ISBN")
TitIdx(1).Name = "Au_ID"
TitIdx(1).Primary = False
TitIdx(1).Unique = False
Set IxFld(1) = TitIdx(1). CreateField("Au_ID")
TitIdx(2).Name = "PubID"
TitIdx(2).Primary = False
TitIdx(2).Unique = False
Set IxFlds(2) = TitIdx(2). CreateField("PubID")
' Append fields to Index object of Titles Table.
For i = 0 to 2
   TitIdx(i).Fields.Append IxFld(i)
Next i
' Append Indexes to Titles TableDef
For i = 0 to 2
   TitTd.Indexes.Append TitIdx(i)
Next i
' Save TableDef definition by appending it to TableDefs
' collection.
MyDB.TableDefs.Append TitTd
```

```
' Create new TableDef for Publishers table.
Set PubTd = MyDB.CreateTableDef("Publishers")
' Add field to PubTd.
Set PubFlds(0) = PubTd.CreateField("PubID", dbLong)
' Make it a counter field.
PubFlds(0).Attributes = dbAutoincrfield
Set PubFlds(1) = PubTd.CreateField("Name", dbText)
PubFlds(1).Size = 50
Set PubFlds(2) = PubTd.CreateField("Company Name", _
dbText)
PubFlds(2).Size = 255
Set PubFlds(3) = MyTd.CreateField("Address", dbText)
PubFlds(3).Size = 50
Set PubFlds(4) = PubTd.CreateField("City", dbText)
PubFlds(4).Size = 20
Set PubFlds(5) = MyTd.CreateField("State", dbText)
PubFlds(5).Size = 10
Set PubFlds(6) = PubTd.CreateField("Zip", dbText)
PubFlds(6).Size = 15
Set PubFlds(7) = PubTd.CreateField("Telephone", dbText)
PubFlds(7).Size = 15
Set PubFlds(8) = PubTd.CreateField("Fax", dbText)
PubFlds(8).Size = 15
Set PubFlds(9) = MyTd.CreateField("Comments", dbText)
PubFlds(9).Size = 50
' Save Field objects by appending to Fields collection.
For i = 0 to 9
   PubTd.Fields.Append PubFld(i)
Next i
' Now add an index.
Set PubIdx = PubTd.CreateIndex("PubID")
PubIdx.Primary = True
PubIdx.Unique = True
Set IxFlds(0) = PubIdx.CreateField("PubID")
PubIdx.Fields.Append IxFlds(0)
PubTd.Indexes.Append PubIdx
' Save TableDef object by appending it to TableDefs
' collection.
MyDB.TableDefs.Append PubTd
MsgBox ("Database Created")
MyDB.Close
End Sub
```

Attaching Tables from External Databases

In addition to defining your own tables, you can attach tables from any of the supported external databases or from another Jet database. When you attach an external table, the connection information is stored in your native Jet database, and the connection is automatically made whenever you open the database. The data itself remains in the external database.

You can use attached tables like any other table in your Visual Basic database, even while they are being shared by other applications, except that you cannot create Recordset objects of the dbOpenTable type. You can also combine operations that include external data from attached tables with data stored in the local database, giving you tremendous flexibility to perform heterogeneous "multiple-database" queries.

It is recommended that you always use attached tables when querying ODBC databases, because it is much faster than opening the remote database directly.

The procedure for attaching an external database table is similar to that for defining and creating a new native table, except that you must specify a SourceTableName and a Connect string when creating the TableDef object, and you can't create or append new Field and Index objects.

▶ To attach an external table to a database

1 Use the CreateTableDef method of the Database object to create a new TableDef object for the attached table. Give the table a new Name that you will call it in the local database. For example, to attach the Customers table from the Nwind.mdb database to the Biblio.mdb database, you could use the following code (assuming Biblio.mdb is open and set to the MyDB database object variable):

```
Set NewTd = MyDB.CreateTableDef("MyCustomers")
```

2 Set the SourceTableName and Connect properties of the TableDef object to specify the external table and database you want to attach:

```
NewTd.SourceTableName = "Customers"
NewTd.Connect = ";DATABASE=C:\Access\Nwind.mdb;"
```

3 Use the Append method to add the new TableDef to the database:

```
MyDB.TableDefs.Append NewTd
```

Depending on the type of external table you're attaching, you may need to provide additional information in the Connect property to specify the ISAM or ODBC driver, Password, User ID, or other required parameters. If you want to query external databases directly, using the dbSqlPassthrough option through attached tables, you must first set the Connect property of the Database object to which the attached .mdb table belongs. For detailed information on the Connect property and other considerations for attaching tables, see Chapter 7, "Accessing External Data."

Modifying a Database

You can use Visual Basic and data access objects to change the structure of your database programmatically.

Modifying the database is very similar to creating it in the first place. In most cases, you use the same Create and Append methods to add objects. You can add new TableDef objects to a database, or add new Field and Index objects to existing tables. You can also delete a TableDef from a database, or delete an Index from a TableDef. Certain restrictions apply to deleting Field objects. For more information, see "Changing or Deleting a Field," later in this chapter.

You can also modify the database using SQL statements. For more information about SQL statements, see Chapter 5, "Writing SQL Queries."

Adding a Table to a Database

To add a table to a database, simply append a new TableDef object to an existing TableDefs collection. For a complete example, refer to instructions earlier in this chapter for creating a new table with fields and indexes. To add a table to a database, you can use the following code:

```
Dim db As Database
Dim NewTd As TableDef    ' Create new TableDef object.
Dim NewFld As Field      ' Create a new Field object.

Set db = DBEngine.Workspaces(0).OpenDatabase _
("Test.mdb")

Set NewTd = db.CreateTableDef("New Table Name")
Set NewFld = NewTd.CreateField("New Field Name", _
dbInteger)

NewTd.Fields.Append NewFld ' Add Field to TableDef.
db.TableDefs.Append NewTd  ' Add TableDef to Database.
db.Close
```

Deleting a Table

To delete a table, you use the Delete method of the TableDefs collection. Deleting a table deletes all the fields, indexes, and data contained within the table, so you should use this method carefully. The following example deletes the Authors table:

```
Db.TableDefs.Delete "Authors"
```

Adding a Field to a Table

You can add a field to an existing table by appending a field to an existing Fields collection. The following code adds two new fields, Address and Phone, to the Authors table.

```
Dim Db As Database
Dim Td As TableDef
Dim Fld As Field

Set Db = DBEngine.Workspaces(0).OpenDatabase _
("C:\Test.mdb")
Set Td = Db.TableDefs("Authors")
' Create the first new field.
Set Fld = Td.CreateField("Address", dbText, 30)
' Append the new Field object to the Fields collection.
Td.Fields.Append Fld
' Reuse the Fld variable to create the second field.

Set Fld = Td.CreateField("Phone", dbText, 25)
' Append to the Fields collection.
Td.Fields.Append Fld
Db.Close
```

Changing or Deleting a Field

You cannot change an individual Field once it is appended to a TableDef. You can delete a Field only if it is not a part of any Index or Relation objects.

To delete an individual Field, use the Delete method of the TableDef object. To change an individual Field, you must add a new TableDef that reflects the desired changes in the fields, then move the data to the new table and delete the old table.

Note To delete an indexed field, you must first delete the index (see "Deleting an Index" later in this chapter). You must also delete any affected Relation objects before you can delete a Field or TableDef object that is part of a relation.

Adding an Index to a Table

You can use the Append method to add a new Index object to an existing Indexes collection. The following example adds a new index to the Authors table:

```
Dim Db As Database, Td As TableDef, NewIdx As Index, _
NewFld As Field

Set Db = DBEngine.Workspaces(0).OpenDatabase _
("C:\Biblio.mdb")
Set Td = Db.TableDefs("Authors")
Set NewIdx = Td.CreateIndex("Address_Index")
NewIdx.Unique = False
Set NewFld = NewIdx.CreateField("Address")

NewIdx.Fields.Append NewFld    ' Add Field to Index
Td.Indexes.Append NewIdx       ' Add Index to TableDef
Db.Close
```

Deleting an Index

The syntax for deleting an index is similar to that used to delete a table. The code in the following example removes the "Address_Index" index from the Indexes collection in the Authors table:

```
Db.TableDefs("Authors").Indexes.Delete "Address_Index"
```

You can't delete an Index that is supporting a Relation.

Database Maintenance

If you are responsible for the maintenance of your database, you will need to perform a number of system-oriented tasks such as compacting the database from time to time, and repairing it if it becomes corrupted.

These tasks include:

- Mapping a database to ascertain its current structure.
- Compacting a database to save disk space and improve performance.
- Repairing a database if it becomes damaged or corrupted.

The following topics discuss these common maintenance tasks, beginning with a method for mapping the current structure of the database.

Mapping a Database

In maintaining your database, it may be helpful to view the underlying structure, or schema, to determine the data access object names and properties. This is called mapping the database. When you have a complete map of your database, you'll find it easier to modify, and to write applications that manipulate the data.

Because data access objects exist as a hierarchy of nested containers, you can map the database by walking through the hierarchy with a series of nested For Each loops. Each element in the collection can be accessed with a statement block of the form.

```
For Each object in collection
   ' Map the properties of the object.
Next object
```

When you map a database, you expose a number of properties that are not fully discussed in this chapter. For more information on these properties, see the *Language Reference* in Books Online for the name of any specific property.

The following example code maps the Biblio.mdb database. In examining the commented code, please note that the Attributes properties are only mapped fully for the TableDef objects, for brevity. If you want to map attributes of the other objects, simply use the code for TableDef attributes as a model and insert the appropriate statements where indicated.

```
Sub MapDatabase ()
   Dim Db As Database, Td As TableDef, Fld As Field
   Dim Idx As Index, Rel As Relation
   Dim i As Integer, n As Integer

   Set Db = DBEngine.Workspaces(0)._
   OpenDatabase("Biblio.mdb")

   ' Map the Database properties.
   Print "DATABASE"
   Print "Name:", Db.Name
   Print "Connect string:", Db.Connect
   Print "Transactions supported?:", Db.Transactions
   Print "Updatable?:", Db.Updatable
   Print "Sort order:", Db.CollatingOrder
   Print "Query time-out:", Db.QueryTimeout
   ' Map the TableDefs
   Print "TABLEDEFS"
   For Each Td in Db.TableDefs
      Print "Name:", Td.Name
      Print "Created:", Td.DateCreated,
      Print "Updated:", Td.LastUpdated,
      If Td.Updatable = True Then
         Print "Updatable",
      Else
         Print "Not Updatable",
      End If
```

```
' Show the TableDef Attributes.
Print Hex$(Td.Attributes)
If (Td.Attributes And dbSystemObject) <> 0 Then
   Print "System Object"
End If
If (Td.Attributes And dbAttachedTable) <> 0 Then
   Print "Attached table"
End If
If (Td.Attributes And dbAttachedODBC) <> 0 Then
   Print "Attached ODBC Table"
End If
If (Td.Attributes And dbAttachExclusive)<> 0 _
Then
   Print "Attached Table opened in exclusive _
   mode"
End If
' Map Fields for each TableDef.
Print "FIELDS"
For Each Fld in Td.Fields
   Print "Name:", Fld.Name
   Print "Type:", Fld.Type
   Print "Size:", Fld.Size
   Print "Attribute Bits:", Hex$(Fld.Attributes)
   Print "Collating Order:", Fld.CollatingOrder
   Print "Ordinal Position:", _
   Fld.OrdinalPosition
   Print "Source Field:", Fld.SourceField
   Print "Source Table:", Fld.SourceTable

   ' Show the Field Attributes here if desired,
   ' as shown
   Print Hex$(Fld.Attributes)
   If (Fld.Attributes And dbSystemObject) <> 0 _
   Then
      Print "System Object"
   End If
Next Fld ' Get the next Field in the TableDef.

' Map Indexes for each TableDef.
Print "INDEXES"
For Each Idx in Td.Indexes
   ' Set the Index variable.
   Set Idx = Td.Indexes(n)
   Print "Name:", Idx.Name
   Print "Clustered:", Idx.Clustered
   Print "Foreign:", Idx.Foreign
   Print "IgnoreNulls:", Idx.IgnoreNulls
   Print "Primary:", Idx.Primary
   Print "Unique:", Idx.Unique
   Print "Required:", Idx.Required
   ' Map the Fields of the Index.
```

```
            For Each Fld in Idx.Fields
               Print "Name:", Fld.Name
               Print "ForeignName:", Fld.ForeignName
            Next Fld ' Get the next Field in the Index.
         Next Idx ' Get the next Index in the TableDef.
      Next Td  ' Get next TableDef in the Database.

      ' Map the Relations.
      Print "RELATIONS"
      For Each Rel in Db.Relations
         Print "Name:", Rel.Name
         Print "Attributes:", Rel.Attributes
         Print "Table:", Rel.Table
         Print "ForeignTable:", Rel.ForeignTable
         ' Map the Fields of the relation
         For Each Fld in Rel.Fields.Count
            Print "Name:", Fld.Name
            Print "ForeignName:", Fld.ForeignName
         Next Fld ' Get the next Field in the Relation.
      Next Rel     ' Get next Relation in the Database.
   End Sub
```

Compacting a Database

To maintain a high state of performance, Visual Basic defers the removal of discarded pages until you shut down the database and compact the discarded pages. This design keeps the interactive performance of your database high at the expense of recoverable disk space. Over time, you'll find that the .mdb database file grows substantially, based on the amount of data you've added and deleted.

You can use the CompactDatabase method of the DBEngine object to copy all data from one database into another, and in the process, organize the data in the resulting database contiguously so that disk space can be recovered. You can compact only Jet databases. A trappable error will be generated if you attempt to use this statement on any open database, or on a database not in the Jet database format.

CompactDatabase also gives you the option of changing the encryption state, version, and locale of a database while it's being compacted. In other words, you can choose to convert an encrypted database to a nonencrypted database, or vice versa. You can also convert from one locale to another or from one version to another.

Before you compact a database, it must be closed. In addition, you should never give the same source and destination names for the database being compacted, because your database will be deleted if the CompactDatabase method is not completed. Compacting cannot be done inside a transaction.

The syntax for the CompactDatabase method is:

DBEngine.CompactDatabase *sourcename, destinationname* [,*locale* [,*options*]]

Argument	Description
sourcename	The *sourcename* argument is the full path and name of the database file you are compacting.
destinationname	The *destinationname* argument is the full path and name of the newly compacted database file. Do not specify the same name for both the source and destination files.
locale	The *locale* argument is the same as the CreateDatabase locale argument.
options	The *options* argument is the same as the CreateDatabase options argument.

For example, the following code compacts a database named Old.mdb and creates a new database with a locale of Spanish and a destination name New.mdb:

```
DBEngine.CompactDatabase "C:\Old.mdb", C:\VB\New.mdb, dbLangSpanish
```

Repairing a Database

If your Jet database is damaged, you can use the RepairDatabase method of the DBEngine object. This method will validate a database that was left invalid by an incomplete data page read/write operation. This kind of corruption can occur when the system isn't shut down normally (such as a power failure). The RepairDatabase method can't fix all possible forms of database corruption, so you should always remember to back up your database files regularly to avoid unrecoverable data loss.

The RepairDatabase method checks all pages in the database for correct linkage, validates all system tables, and validates all indexes. Upon successful completion, the database is rendered "safe," and any pages that can't be salvaged (such as invalid references to other pages) are discarded.

The RepairDatabase method takes only one argument, which is the name of the database file you want to repair. For example, to repair the Biblio.mdb database, you could use the following code:

```
DBEngine.RepairDatabase "Biblio.mdb"
```

When a database is repaired, it may increase in size, because the process of creating indexes may leave some deleted pages in the database. It's always a good idea to run the CompactDatabase method after any repair to eliminate unnecessary pages.

CHAPTER 3

Working with Records and Fields

The Microsoft Jet database engine supports a rich set of Data Access Objects (DAO) features for organizing, sorting, searching, updating, adding, and deleting data. The Recordset object alone provides 24 methods and 26 properties that give you a great deal of control over records in a database. With the Recordset object's Fields collection and a number of Field object properties and methods, you can manipulate data at the field level. This chapter describes in detail how to manipulate records and fields using the DAO Recordset and Field objects.

Using Recordset Objects

A Recordset object represents the records in a base table or the records that result from running a query. You use Recordset objects to manipulate the data in a database at the record level.

Note You use Field objects to manipulate data in a database at the field level. For more information, see "Using Field Objects" later in this chapter.

The five types of Recordset objects—table, dynaset, snapshot, dynamic, and forward-only — differ from each other in significant ways:

- A *table-type* Recordset object refers to a local table in the current database or to an external database created with Microsoft Jet. When you create a table-type recordset, the database engine opens the actual table, and your subsequent data manipulations operate directly on base-table data. You can open a table-type recordset against only one table; you cannot open it against a join or union query.

  You can index a table-type Recordset object by using an index created for the underlying table. This provides much faster sorting and searching than is possible with the other types of Recordset objects. The Seek method is faster than the Find methods for locating specific records.

- A *dynaset-type* Recordset object refers to either local or linked tables, or to the result of a row-returning query. It's actually a set of references to records in one or more tables. With a dynaset, you can extract and update data from more than one table, including linked tables from other databases. Heterogeneous updatable joins are a unique feature of dynasets—they enable you to use updatable join queries against tables in different types of databases.

  A dynaset and its underlying tables update each other. Record changes in the dynaset are also made in the underlying table, and when other users change records in the underlying table while the dynaset is open, those record changes are reflected in the dynaset. The dynaset is the most flexible and powerful type of recordset, although searches and other manipulations may not execute as quickly as with a table-type Recordset.

- A *snapshot-type* Recordset object contains a fixed copy of the data as it exists at the time the snapshot is created. A snapshot derived from a Microsoft Jet data source can't be updated. Some snapshots derived from Open Database Connectivity (ODBC) data sources can be updated, depending on the capabilities of the back-end database.

  A snapshot creates less processing overhead than the dynaset- and table-type Recordset objects, so it may execute queries and return data faster, especially when working with ODBC data sources. Note that for .mdb files, pointers represent data in Memo and Long Binary fields in a snapshot.

- A *forward-only–type* Recordset object, sometimes referred to as a "forward-scrolling snapshot" or "forward-only snapshot," provides a subset of the capabilities of a snapshot. It offers the least functionality of any Recordset object, but usually provides the greatest speed. Like snapshots, forward-only-type Recordset objects derived from Microsoft Jet data sources aren't updatable. Additionally, forward-only snapshots only let you move in a forward direction through the records. Recordset objects of this type cannot be cloned and only support the Move and MoveNext methods.

- A *dynamic-type* Recordset object is a query result set from one or more base tables in which you can add, change, or delete records from a row-returning query. Further, records other users add, delete, or edit in the base tables also appear in your recordset. This type corresponds to an ODBC dynamic cursor.

  For More Information For information about the dynamic-type Recordset object, see Chapter 9, "Developing Client/Server Applications."

Note Because a snapshot stores a complete copy of all the records in a table, a snapshot may perform more slowly than a dynaset if the number of records is large. To determine whether a snapshot or dynaset is faster, you can open the recordset as a dynaset and then open it as a snapshot.

The type of recordset you use depends on what you want to do and whether you want to change or simply view the data. For example, if you must sort the data or work with indexes, use a table. Because table-type Recordset objects are indexed, they also provide the fastest way to locate data. If you want to be able to update a set of records selected by a query, use a dynaset. If the table-type Recordset is unavailable in a specific situation and you only need to scan through a set of records, using a forward-only snapshot may be faster.

Generally, if a table-type Recordset object is available, it almost always results in the best performance.

Note In this chapter, the terms table, snapshot, and dynaset are often used for the sake of simplicity. However, keep in mind that these are all types of Recordset objects. For example, the term dynaset refers to a dynaset-type Recordset object, not the obsolete DAO Dynaset object.

Creating a Recordset Variable

Use the OpenRecordset method to create a Recordset object variable. First, you must declare a variable of type Recordset, and then set the variable to the object returned by the OpenRecordset method.

The OpenRecordset method is available from Database, Connection, TableDef, QueryDef, and existing Recordset objects. The syntax of the OpenRecordset method for Connection and Database objects is:

Set *variable* = *database*.**OpenRecordset** (*source* [, *type* [, *options*, [*lockedits*]]])

The syntax of the OpenRecordset method for all other types of objects is:

Set *variable* = *object*.**OpenRecordset** ([*type* [, *options* [,*lockedits*]])

The *variable* argument is the name of the new Recordset object.

The *database* argument is the name of an open Database or Connection object from which you're creating the new Recordset object.

The *object* argument is a TableDef, QueryDef, or existing Recordset object from which you're creating the new Recordset object.

The *source* argument specifies the source of the records for the new Recordset object. The value of *source* becomes the value of the resulting Recordset object's Name property. When you create a new Recordset object from a Connection or Database

object, the *source* argument is an existing TableDef or QueryDef object in the database or a valid row-returning SQL query or statement. When you create a new Recordset object from a TableDef, QueryDef, or existing Recordset object, the object itself provides the data source for the new recordset.

The *type* argument is an intrinsic constant that specifies the kind of recordset you want to create. You can use the following constants:

- dbOpenTable
- dbOpenDynaset
- dbOpenSnapshot
- dbOpenForwardOnly
- dbOpenDynamic

The type dbOpenForwardOnly replaces the dbForwardOnly option that was available in previous versions of DAO. You can still use the dbForwardOnly option, but it is provided only for backward compatibility.

The next sections discuss the *type, options,* and *lockedits* arguments in detail.

Default Recordset Types

DAO automatically bases the default recordset type on the recordset data source and how you open the recordset. Therefore, you don't need to specify a recordset type. However, you can specify a *type* argument in the OpenRecordset method to override the default type.

The following list describes the available types and the default type, depending on how you open the Recordset object.

- Using the OpenRecordset method with a Database object:

  ```
  Set rstNew = dbs.OpenRecordset("Data Source")
  ```

  If `Data Source` is a local table in the database, table-, dynaset-, snapshot-, and dynaset-type Recordset objects are available, and the default recordset type is table. If `Data Source` is anything else, only dynaset- and snapshot-type Recordset objects are available, and the dynaset type is the default.

- Using the OpenRecordset method with a TableDef object:

  ```
  Set rstNew = tdfTableData.OpenRecordset
  ```

  If `tdfTableData` refers to a table in a Microsoft Jet database (.mdb) or to an ISAM database opened directly, then all five types are available and the table-type recordset is the default type. If `tdfTableData` is in an ODBC database or is a linked table in an external database, only dynaset- and snapshot-type Recordset objects are available, and the dynaset type is the default.

Using the OpenRecordset method with a QueryDef object:

```
Set rstNew = qdfQueryData.OpenRecordset
```

Only dynaset- and snapshot-type Recordset objects are available, and the dynaset type is the default.

- Using the OpenRecordset method with an existing Recordset object:

  ```
  Set rstNew = rstExisting.OpenRecordset
  ```

  Only dynaset- and snapshot-type Recordset objects are available. The default is the type of the existing recordset, in this case, the type of `rstExisting`.

For More Information For information about using the OpenRecordset method with a Connection object, see Chapter 9, "Developing Client/Server Applications."

Specifying OpenRecordset Options

You can use the *options* argument of the OpenRecordset method to specify a number of other features for a Recordset object. You can use the constants in the following table.

Constant	Description
dbAppendOnly	Allows users to append new records to the recordset, but prevents them from editing or deleting existing records (dynaset only). This is useful in applications that collect and archive data.
dbReadOnly	Prevents users from making changes to the recordset (Microsoft Jet workspaces only). The dbReadOnly constant in the lockedits argument replaces this option, which is provided only for backward compatibility.
dbSeeChanges	Generates a run-time error if one user is changing data that another user is editing (Microsoft Jet dynaset-type recordset only). This is useful in applications where multiple users have simultaneous read/write access to the same data.
dbDenyWrite	Prevents other users from modifying or adding records (Microsoft Jet Recordset objects only).

Constant	Description
dbDenyRead	Prevents other users from reading data in a table (Microsoft Jet table-type Recordset only).
dbForwardOnly	Creates a forward-only snapshot. It is provided only for backward compatibility and is available only for Microsoft Jet data sources. Use the dbOpenForwardOnly constant in the *type* argument instead of using this option.
dbSQLPassThrough	Passes an SQL statement to an ODBC database for processing, if *source* is an SQL statement (snapshot only).
dbConsistent	Allows only consistent updates (Microsoft Jet dynaset-type and snapshot-type Recordset objects only). You can't use this in conjunction with dbInconsistent, and dbConsistent is the default.
dbInconsistent	Allows inconsistent updates (Microsoft Jet dynaset-type and snapshot-type Recordset objects only).

For More Information For more information about inconsistent updates, see "Inconsistent Updates" later in this chapter.

Specifying Recordset Locking Options

You can use the *lockedits* argument to control locking for a recordset. Use the following constants.

Constant	Description
dbReadOnly	Prevents users from making changes to the recordset. This constant replaces the dbReadOnly constant that was used in the *options* argument in previous versions of DAO.
DbPessimistic	Uses pessimistic locking to determine how changes are made to the recordset in a multiuser environment.
DbOptimistic	Uses optimistic locking to determine how changes are made to the recordset in a multiuser environment.
DbOptimisticValue	Uses optimistic concurrency based on row values instead of row IDs. Used for ODBCDirect data sources only.

The default value for ODBCDirect data sources is dbReadOnly. The default value for Microsoft Jet databases is dbPessimistic. For Microsoft Jet databases, using dbPessimistic or dbOptimistic only presets the value of the Recordset object's LockEdits property.

Important Setting both the *lockedits* argument and the *options* argument to dbReadOnly causes a run-time error.

For More Information For more information about optimistic and pessimistic locking, see Chapter 8, "Developing Multiuser Applications."

Creating a Recordset from a Table

The method you use to create a Recordset object from a base table depends on whether the table is local to the current database or is a linked table in another database.

Creating a Recordset from a Table in a Local Microsoft Jet Database

The following code uses the OpenRecordset method to create a table-type Recordset object for a table:

```
Dim dbs As Database, rstCustomers As Recordset

Set dbs = OpenDatabase("Northwind.mdb")
Set rstCustomers = dbs.OpenRecordset("Customers")
```

Notice that you don't need to use the dbOpenTable constant to create a table-type recordset. If you omit the *type* constant, DAO uses the most appropriate type of Recordset object for the data source and the object you use to create the recordset. Because the table-type recordset is available when you open a recordset from a local table, DAO uses it.

Creating a Recordset from a Linked Table in a Different Database Format

The following code creates a dynaset-type Recordset object for a linked Paradox 3.x table. Because the table type isn't available when you open a recordset from a linked table in a non-Microsoft Jet database, DAO uses the next most efficient type, opening a dynaset-type recordset.

```
Dim dbs As Database
Dim tdfNonJetLinked As TableDef
Dim rstTableData As Recordset

' Open the database and create a TableDef.
Set dbs = OpenDatabase("Northwind.mdb")
Set tdfNonJetLinked = dbs.CreateTableDef("PDXAuthor")

' Connect to the Paradox table Author in the database
' C:\PDX\Publish.
tdfNonJetLinked.Connect = _
   "Paradox 3.X;DATABASE=C:\PDX\Publish"
tdfNonJetLinked.SourceTableName = "Author"

' Link the table.
dbs.TableDefs.Append tdfNonJetLinked

' Create a dynaset-type recordset for the table.
Set rstTableData = tdfNonJetLinked.OpenRecordset()
```

You can also open a Paradox table directly by first opening the Paradox database.

Note In ODBCDirect data sources, you can also use the OpenRecordset method on the Connection object. For more information, see Chapter 9, "Developing Client/Server Applications."

For More Information For more information about connecting to data in external databases, see Chapter 7, "Accessing External Data."

Using an Index on a Table-Type Recordset

To order records in a table-type Recordset object, set its Index property.

The following code uses an index on a table-type recordset based on the Employees table. It returns the hire date for a specified employee and assumes the name of the index for the EmployeeID field is "PrimaryKey".

```
Function GetHireDate(lngEmpID As Long, varHireDate _
   As Variant) As Integer

   Dim rstEmployees As Recordset, dbs As Database

   On Error GoTo ErrorHandler
   Set dbs = OpenDatabase("Northwind.mdb")
   Set rstEmployees = dbs.OpenRecordset("Employees", _
      dbOpenTable)

   ' Set the index and find the specified Employee ID
   rstEmployees.Index = "PrimaryKey"
   rstEmployees.Seek "=", lngEmpID

   If rstEmployees.NoMatch Then
      varHireDate = Null
      GetHireDate = ERR_NOMATCH
      Exit Function
   Else
      varHireDate = rstEmployees!HireDate
      GetHireDate = SUCCESS
      Exit Function
   End If

ErrorHandler:
Select Case Err
   Case 0
      Exit Function
   Case Else
      varHireDate = Null
      GetHireDate = FAILED
      MsgBox "Error " & Err & ": " & Error, _
         vbOKOnly, "ERROR"
      Exit Function
End Select

End Function
```

You must set the Index property before using the Seek method. If you set the Index property to an index that doesn't exist, a trappable error occurs. If you want to sort records according to an index that doesn't exist, either create the index first, or create a dynaset- or snapshot-type recordset using a query that returns records in a specified order.

Creating a Recordset from a Query

You can also create a Recordset object based on a stored select query. In the following example, Current Product List is an existing select query stored in the current database.

```
Dim dbs As Database, rstProducts As Recordset

Set dbs = OpenDatabase("Northwind.mdb")
Set rstProducts = dbs.OpenRecordset _
   ("Current Product List")
```

If a stored select query doesn't already exist, the OpenRecordset method also accepts an SQL string instead of the name of a query. The previous example can be rewritten as follows:

```
Dim dbs As Database, rstProducts As Recordset
Dim strQuerySQL As String

Set dbs = OpenDatabase("Northwind.mdb")
strQuerySQL = "SELECT * FROM Products " _
   & "WHERE Discontinued = No " _
   & "ORDER BY ProductName;"
Set rstProducts = dbs.OpenRecordset(strQuerySQL)
```

The disadvantage of this approach is that the query string must be compiled each time it's executed, whereas the stored query is compiled the first time it's saved, which usually results in slightly better performance.

Note When you create a Recordset object in Microsoft Jet using an SQL string or a stored query, your code doesn't continue running until the query returns the first row in the recordset. You may want to display a message in the status bar while the query is running. With ODBCDirect, you can specify dbRunAsync, which enables your code to continue processing other statements while the query runs in the background.

For More Information For more information about SQL statements, see Chapter 5, "Writing SQL Queries."

Sorting and Filtering Records

To ensure records will appear in a specific order, you can open a table-type Recordset object and set its Index property. For example, you may want to either view invoices arranged by increasing invoice number, or retrieve employee records in alphabetic order by their last names.

To sort non-table recordset data, use an SQL ORDER BY clause in the query that constructs the recordset. You can specify an SQL string when you create a QueryDef object, when you create a stored query in a database, or when you use the OpenRecordset method.

With any type of recordset, use an SQL WHERE clause in the original query to filter data (to restrict the result set to records that meet some criteria).

The following code opens a dynaset-type Recordset object. It uses WHERE and ORDER BY clauses to retrieve, filter, and sort records.

```
Dim dbs As Database, rstManagers As Recordset

Set dbs = OpenDatabase("Northwind.mdb")
Set rstManagers = dbs.OpenRecordset("SELECT " _
   & "FirstName, LastName FROM Employees WHERE " _
   & "Title = 'Sales Manager' " _
   & "ORDER BY LastName")
```

One drawback of running an SQL query in an OpenRecordset method is that it has to be recompiled every time you run it. To improve performance with frequently used queries, first create a stored query using the same SQL statement, and then open a Recordset object against the query, as shown in the following code:

```
Dim dbs As Database
Dim rstSalesReps As Recordset
Dim qdf As QueryDef

Set dbs = OpenDatabase("Northwind.mdb")

Set qdf = dbs.CreateQueryDef("SalesRepQuery")
qdf.SQL = "SELECT * FROM Employees WHERE " _
   & "Title = 'Sales Representative'"

Set rstSalesReps = dbs.OpenRecordset("SalesRepQuery")
```

For even greater flexibility and control at run time, you can use query parameters to determine the sort order and filter criteria. For more information, see "Using Parameter Queries" later in this chapter.

Recreating a Query from a Recordset

If you open a Recordset object from a QueryDef object, you can use CopyQueryDef to create a new QueryDef that is a duplicate of the QueryDef used to create the recordset. This is useful in situations where a Recordset variable created from a QueryDef is passed to a function, and the function must recreate the SQL equivalent of the query and possibly modify it.

Modifying a Query from a Recordset

You can use the Requery method on a dynaset- or snapshot-type Recordset when you want to re-execute the underlying query after changing a parameter. This is more convenient than opening a new Recordset, and it runs faster.

The following code creates a Recordset object and passes it to a function that uses the CopyQueryDef method to extract the equivalent SQL string. It then prompts the user to add an additional constraint clause to the query.

```
Sub AddQuery()

   Dim dbs As Database
   Dim qdf As QueryDef
   Dim rstSalesReps As Recordset

   Set dbs = OpenDatabase("Northwind.mdb")

   Set qdf = dbs.CreateQueryDef("SalesRepQuery")
   qdf.SQL = "SELECT * FROM Employees WHERE " _
      & "Title = 'Sales Representative'"

   Set rstSalesReps = qdf.OpenRecordset()

   ' Call the function to add a constraint.
   AddQueryFilter rstSalesReps

   ' Return database to original.
   dbs.QueryDefs.Delete "SalesRepQuery"

End Sub
```

```
Function AddQueryFilter(rst As Recordset)

   Dim qdf As QueryDef
   Dim strNewFilter As String, strRightSQL As String

   Set qdf = rst.CopyQueryDef

   ' Try "LastName LIKE 'D*'"
   strNewFilter = InputBox("Enter new constraint")

   strRightSQL = Right(qdf.SQL, 1)
   ' Strip characters from the end of the query,
   ' as needed.
   Do While strRightSQL = " " Or strRightSQL = ";" _
      Or strRightSQL = vbCR Or _
      strRightSQL = vbLF
      qdf.SQL = Left(qdf.SQL, Len(qdf.SQL) - 1)
      strRightSQL = Right(qdf.SQL, 1)
   Loop
   qdf.SQL = qdf.SQL & " AND " & strNewFilter & ";"

   rst.Requery qdf       ' Requery the recordset.
   rst.MoveLast          ' Populate the recordset.

   ' "Lastname LIKE 'D*'" should return 2 records.
   MsgBox "Number returned = " & rst.RecordCount

End Function
```

Note To use the Requery method, the Recordset object's Restartable property must be True. The Restartable property is always True when the recordset is created from a query other than a crosstab query against tables in a Microsoft Jet database. You can't restart SQL pass-through queries. You may or may not be able to restart queries against linked tables in another database format.

The Sort and Filter Properties

Another approach to sorting and filtering Recordset objects is to set the Sort and Filter properties on an existing recordset, and then open a new recordset from the existing one. However, this is usually much slower than just including the sort and filter criteria in the original query or changing the query parameters and re-executing it with the Requery method. The Sort and Filter properties are useful when you want to allow a user to sort or restrict a result set, but the original data source is unavailable for a new query—for example, when a Recordset object variable is passed to a function, and the function must reorder records or restrict the records in the set. With this approach, performance is likely to be slow if more than 100 records are in the Recordset. Using the CopyQueryDef method described in the previous section is preferable.

Moving Through a Recordset

When you refer to the fields in a recordset, you obtain values from the record at the current position, called the *current record*. However, the current position can also be immediately before the first record in a recordset or immediately after the last record. In certain circumstances, the current position is undefined.

You can use the Move methods to loop through the records in a recordset:

- The MoveFirst method moves to the first record.
- The MoveLast method moves to the last record.
- The MoveNext method moves to the next record.
- The MovePrevious method moves to the previous record.
- The Move [*n*] method moves forward or backward a specified number [*n*] of records.

You can use each of these methods on table-, dynaset-, and snapshot-type Recordset objects. On a forward-only-type recordset, you can use only the MoveNext and Move methods. If you use the Move method on a forward-only-type recordset, the argument specifying the number of rows to move must be a positive integer.

The following code changes the job title of all sales representatives in a table called Employees. After opening the table, the code sets the index, which makes it necessary to use the MoveFirst method to locate the first record. For each record satisfying the title condition, the example changes the title and saves the change with the Update method. It uses the MoveNext method to move to the next record.

The following code opens a recordset containing all Employees table records that have a Null value in the ReportsTo field. Using the MoveNext method, it moves through the records in the recordset and updates their records to indicate that they are temporary employees. The Update method saves changes in the table.

```
Function UpdateEmployees()

   Dim dbs As Database
   Dim rstEmployees As Recordset
   Dim strQuery As String

   On Error GoTo ErrorHandler
```

```
    Set dbs = OpenDatabase("Northwind.mdb")

    ' Open a recordset on all records in the
    ' Employees table that have a Null value in the
    ' [ReportsTo] field.

    strQuery = "SELECT * FROM Employees WHERE " _
       & "ReportsTo IS NULL;"

    Set rstEmployees = dbs.OpenRecordset(strQuery, _
       dbOpenDynaset)

    ' If the recordset is empty, exit
    If rstEmployees.EOF Then Exit Function

    With rstEmployees
       Do Until .EOF
          .Edit
          ![ReportsTo] = 5
          ![Title] = "Temporary"
          ![Notes] = rstEmployees![Notes] _
             & "--Hired from Temp Service--"
          .Update
          .MoveNext
       Loop
       .Close
    End With

ErrorHandler:
Select Case Err
    Case 0
       Exit Function
    Case Else
       MsgBox "Error " & Err & ": " & _
          Error, vbOKOnly, "ERROR"
       Exit Function
End Select

End Function
```

Note The previous example illustrates the MoveNext method, but it would be much faster to perform this bulk operation with an SQL UPDATE query, especially when the number of records to update is large.

You can use the Data control to browse and display records in a Recordset object and to edit data in records as they are displayed. For more information about the Data control, see Chapter 14, "Accessing Data" in the *Programmer's Guide*.

Detecting the Limits of a Recordset

If you try to move too far in one direction in a recordset, a run-time error occurs. For example, if you try to use the MoveNext method when you're already beyond the end of the recordset, a trappable error occurs. For this reason, it's helpful to know the limits of the Recordset object.

The BOF property indicates whether the current position is at the beginning of the recordset. If BOF is True, the current position is before the first record in the recordset. The BOF property is also True if there are no records in the recordset when it's opened. Similarly, the EOF property is True if the current position is after the last record in the recordset or if there are no records.

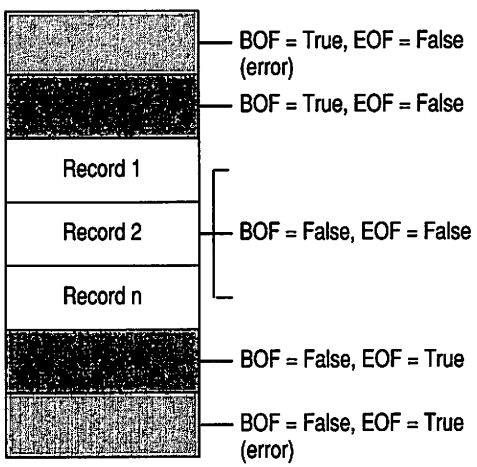
The following code uses the BOF and EOF properties to detect the beginning and end of a recordset. This code fragment creates a table-type recordset based on the Orders table. It moves through the records, first from the beginning of the recordset to the end, and then from the end of the recordset to the beginning.

```
Dim dbs As Database, rstOrders As Recordset

Set dbs = OpenDatabase("Northwind.mdb")
Set rstOrders = dbs.OpenRecordset _
   ("Orders", dbOpenTable)

Do Until rstOrders.EOF
   ' ...Manipulate data...
   rstOrders.MoveNext' Move to the next record.
Loop

rstOrders.MoveLast    ' Move to the last record.

' Do until beginning of file.
Do Until rstOrders.BOF

   ' ...Manipulate data...

   ' Move to the previous record.
   rstOrders.MovePrevious
Loop

rstOrders.Close    ' Close the recordset.
dbs.Close
```

Notice that there's no current record immediately following the first loop. The BOF and EOF properties both have the following characteristics.

- If the recordset contains no records when you open it, both BOF and EOF are True.
- When BOF or EOF is True, the property remains True until you move to an existing record, at which time the value of BOF or EOF becomes False.
- When BOF or EOF is False, and the only record in a recordset is deleted, the property remains False until you attempt to move to another record, at which time both BOF and EOF become True.
- At the moment you create or open a recordset that contains at least one record, the first record is the current record, and both BOF and EOF are False.
- If the first record is the current record when you use the MovePrevious method, BOF is set to True. If you use MovePrevious while BOF is True, a run-time error occurs. When this happens, BOF remains True and there is no current record.
- Similarly, moving past the last record in the recordset changes the value of the EOF property to True. If you use the MoveNext method while EOF is True, a run-time error occurs. When this happens, EOF remains True and there is no current record.

Relationship between the current record position and the BOF and EOF properties



For More Information For more information, see "BOF Property" in the *Language Reference* in Books Online.

Counting the Number of Records in a Recordset

You may want to know the number of records in a Recordset object. For example, you may want to create a form that shows how many records are in each of the tables in a database. Or you may want to change the appearance of a form or report based on the number of records it includes.

The RecordCount property contains the number of records in a table-type recordset or the total number of records accessed in a snapshot- or dynaset-type Recordset object. A recordset with no records has a RecordCount value of 0.

Note The RecordCount value equals the number of records that have actually been accessed. For example, when you first create a dynaset or snapshot, you have accessed (or "visited") only one record. If you check the RecordCount property immediately after creating the dynaset or snapshot (assuming it has at least one record), the value is 1. To visit all the records, use the MoveLast method immediately after opening the recordset. If you then need to return to the first record, use the MoveFirst method. Visiting all records isn't done automatically for dynasets or snapshots because it may be slow, especially for large recordsets.

When you open a table-type Recordset object, you effectively visit all of the records in the underlying table, and RecordCount totals the number of records in the table as soon as the recordset is opened. Canceled transactions may make the RecordCount value out of date in some multiuser situations. Compacting the database restores the table's record count to the correct value.

The following code creates a snapshot-type recordset from a specified SQL SELECT statement, and then returns the number of records in the recordset.

```
Function RecCount(strSQL As String, strDB As String) _      As Long

   Dim rstCount As Recordset
   Dim dbs As Database

   On Error GoTo ErrorHandler
   Set dbs = OpenDatabase(strDB)

   Set rstCount = dbs.OpenRecordset(strSQL, _
      dbOpenSnapshot)

   ' If there are no records, return 0 and exit:

   If rstCount.EOF Then
      rstCount.Close
      RecCount = 0
      Exit Function
   Else
      rstCount.MoveLast
      RecCount = rstCount.RecordCount
      rstCount.Close
      Exit Function
   End If
```

```
ErrorHandler:
Select Case Err
   Case 0
      Exit Function
   Case Else
      MsgBox "Error " & Err & ": " & _
         Error, vbOKOnly, "ERROR"
      Exit Function
End Select

End Function
```

As your application deletes records in a dynaset-type recordset, the value of the RecordCount property decreases. However, in a multiuser environment, when other users delete records, changes aren't reflected in the value of RecordCount until the current record is positioned on a deleted record. At that time, the setting of the RecordCount property decreases by one. To set the RecordCount property to the current total number of records in the recordset, use the Requery method on a recordset, followed by the MoveLast method.

A snapshot-type recordset is static and its RecordCount value doesn't change when you add or delete records in the snapshot's underlying table.

Finding the Current Position in a Recordset

In some situations, you need to determine how far through a recordset you have moved the current record position, and perhaps indicate the current record position to a user. For example, you may want to indicate the current position on a dial, meter, or similar type of control. Two properties indicate the current position: the AbsolutePosition property and the PercentPosition property.

The AbsolutePosition property value is the position of the current record relative to 0. However, don't think of this property as a record number; if the current record is undefined, AbsolutePosition returns –1. Further, there is no guarantee that records will appear in the same order every time the recordset is accessed.

The PercentPosition property expresses the current position as a percentage of the total number of records indicated by the RecordCount property. Because the RecordCount property doesn't reflect the total number of records in the recordset until the recordset has been fully populated, the PercentPosition property only reflects the

current record position as a percentage of the number of records that have been accessed since the recordset was opened. To make sure that the PercentPosition property reflects the current record position relative to the entire recordset, use the MoveLast and MoveFirst methods immediately after opening the recordset. This fully populates the recordset before you attempt to use the PercentPosition property. If you have a large result set, using the MoveLast method may take a long time for non-table-type Recordset objects.

Important The PercentPosition property is only an approximation, and you shouldn't use it as a critical parameter. Use this property for driving an indicator, such as a "percent complete" control, or a similar indicator that marks a user's progress while moving though a set of records.

The following code moves through the Employees table and appends the text "Senior Staff" to the Notes field if the hire date of the employee is before Jan. 1, 1993. The SysCmd function is used to display a progress bar showing the percentage of the table that's been processed.

```
Function PercentPos()

   Dim dbs As Database, strMsg As String
   Dim rstEmployees As Recordset, intRet As Integer
   Dim strQuery As String
   Dim varReturn As Variant

   On Error GoTo ErrorHandler

   strQuery = "SELECT * FROM Employees;"
   Set dbs = OpenDatabase("Northwind.mdb")
   Set rstEmployees = dbs.OpenRecordset _
      (strQuery, dbOpenDynaset)

   ' If no employee records, exit.
   If rstEmployees.EOF Then Exit Function

   With rstEmployees
      strMsg = "Processing Employees table..."
      intRet = SysCmd(acSysCmdInitMeter, strMsg, 100)
```

```
        Do Until .EOF
           If !HireDate < #1/1/93# Then
              .Edit
              !Notes = !Notes & ";" & "Senior Staff"
              .Update
           End If

           If .PercentPosition <> 0 Then
              intRet = SysCmd(acSysCmdUpdateMeter, _
                 .PercentPosition)
           End If
           .MoveNext
        Loop

        .Close

     End With
     intRet = SysCmd(acSysCmdRemoveMeter)
     dbs.Close

ErrorHandler:
Select Case Err
   Case 0
      Exit Function
   Case Else
      MsgBox "Error " & Err & ": " & _
         Error, vbOKOnly, "ERROR"
      ' Clear progress meter.
      varReturn = SysCmd(acSysCmdSetStatus, " ")
   dbs.Close
Exit Function
End Select

End Function
```

Finding a Specific Record

The previous section, "Finding the Current Position in a recordset," explores ways you can use the MoveFirst, MoveLast, MovePrevious, and MoveNextmethods to loop through a recordset. However, it's usually more efficient to search for a specific record.

For example, you may want to find a particular employee based on an employee number, or you may want to find all of the detail records that belong to a specific order. In these cases, looping through all of the employee or order detail records could be time consuming. Instead, you can use the Seek method to locate records with table-type Recordset objects, and the Find methods with dynaset- and snapshot-type Recordset objects.

Finding a Record in a Table-Type Recordset

Use the Seek method to locate a record in a table-type recordset. To locate a record in a dynaset or snapshot, use one of the Find methods described in the next section.

Note Because the forward-only-type recordset doesn't support the Seek method or any of the Find methods, you cannot search for records in a forward-only-type recordset.

When you use the Seek method to locate a record, Microsoft Jet uses the table's current index, as defined by the Index property.

The syntax for the Seek method is:

table.**Seek** *comparison, key1, key2 ...*

The *table* argument is a table-type Recordset object you're searching through. The Seek method accepts a number of arguments, the first of which is *comparison*, a string that determines the kind of comparison that is being performed. The following table lists the comparison strings you can use with the Seek method.

Comparison string	Description
"="	Equal to the specified key values.
">="	Greater than or equal to the specified key values.
">"	Greater than the specified key values.
"<="	Less than or equal to the specified key values.
"<"	Less than the specified key values.

The *keyn* arguments are a series of one or more values that correspond to the field or fields that make up the current index of the recordset. Microsoft Jet compares these values to values in the corresponding fields of the Recordset object's records.

The following code opens a table-type recordset on the Suppliers table, and uses the Seek method to return a supplier ID when the supplier's company is specified. It returns Null if the supplier could not be found.

```
Public Function GetSupplierID(strCompanyName _
   As String) As Variant

   Dim dbs As Database, rstSuppliers As Recordset
```

```
    On Error GoTo ErrorHandler

    Set dbs = OpenDatabase("Northwind.mdb")
    Set rstSuppliers = dbs.OpenRecordset("Suppliers", _
       dbOpenTable)
    rstSuppliers.Index = "CompanyName"
    rstSuppliers.Seek "=", strCompanyName

    If rstSuppliers.NoMatch Then
       GetSupplierID = Null
    Else
       GetSupplierID = rstSuppliers!SupplierID
    End If
    rstSuppliers.Close
    dbs.Close

ErrorHandler:
Select Case Err
    Case 0
       Exit Function
    Case Else
       MsgBox "Error " & Err & ": " & _
       Error, vbOKOnly, "ERROR"
       Exit Function
End Select

End Function
```

The Seek method always searches for records starting at the beginning of the recordset. If you use the Seek method (with the same arguments) more than once on the same recordset, it will find the same record.

You can use the NoMatch property on the Recordset object to test whether a record matching the search criteria was found. If the record matching the criteria was found, NoMatch will be False; otherwise it will be True. If you use the Seek method on a table-type Recordset object without first setting the current index, a run-time error occurs.

The next example is a function that uses the Seek method to locate a record on a table whose primary key is defined by two fields.

```
Function GetFirstPrice(lngOrderID As Long, _
    lngProductID As Long) As Variant
    Dim dbs As Database, rstOrderDetails As Recordset

    On Error GoTo ErrorHandler

    Set dbs = OpenDatabase("Northwind.mdb")
    Set rstOrderDetails = dbs.OpenRecordset _
       ("Order Details", dbOpenTable)
    rstOrderDetails.Index = "PrimaryKey"
    rstOrderDetails.Seek "=", lngOrderID, lngProductID
```

```
    If rstOrderDetails.NoMatch Then
        GetFirstPrice = Null
    Else
        GetFirstPrice = rstOrderDetails!UnitPrice
    End If
    rstOrderDetails.Close

ErrorHandler:
Select Case Err
    Case 0
        Exit Function
    Case Else
        MsgBox "Error " & Err & ": " & Error, _
            vbOKOnly, "ERROR"
        Exit Function
End Select

End Function
```

In this example, the table's primary key consists of the two fields, OrderID and ProductID. When you call the GetFirstPrice function with a valid (existing) combination of OrderID and ProductID field values, the function returns the unit price from the found record. If the combination of field values you want isn't found in the table, the function returns Null.

If the current index is a multiple-field index, you can omit trailing key values, which are treated as Null. That is, you can leave off any number of key values from the end of a Seek method's *key* argument, but not from the beginning or the middle. However, if you don't specify all values in the index, you can use only the ">" or "<" comparison operator with the Seek method.

Finding a Record in a Dynaset- or Snapshot-Type Recordset

You can use the following methods to locate a record in a dynaset- or snapshot-type recordset. DAO provides four Find methods.

- The FindFirst method finds the first record satisfying the specified criteria.
- The FindLast method finds the last record satisfying the specified criteria.
- The FindNext method finds the next record satisfying the specified criteria.
- The FindPrevious method finds the previous record satisfying the specified criteria.

When you use the Find methods, you specify the search criteria, typically an expression equating a field name with a specific value.

Note To locate a record in a table-type recordset, use the Seek method, which is described in the previous section.

Note To add, edit, or delete a record, there must be a unique index on the record in the underlying data source. If not, a "Permission denied" error will occur on the AddNew, Delete, or Edit method call in a Microsoft Jet workspace, or an "Invalid argument" error will occur on the Update call in an ODBCDirect workspace.

The following function finds orders in the Orders table that have no corresponding records in the OrderDetails table and adds the OrderID to the Long array `lngOrderID()`.

```
Function FindOrders(lngOrderID() As Long)

   Dim dbs As Database, rstOrders As Recordset
   Dim strQuery As String, intIndex As Integer
   Dim rstOrderDetails As Recordset

   On Error GoTo ErrorHandler

   Set dbs = OpenDatabase("Northwind.mdb")

   ' Open recordsets on the Orders and OrderDetails
   ' tables. If no records, exit the function.

   strQuery = "SELECT * FROM Orders ORDER BY OrderID;"
   Set rstOrders = dbs.OpenRecordset(strQuery, _
      dbOpenSnapshot)
   If rstOrders.EOF Then Exit Function

   strQuery = "SELECT * FROM OrderDetails ORDER BY "
      & " OrderID;"
   Set rstOrderDetails = dbs.OpenRecordset _
      (strQuery, dbOpenSnapshot)

   ' For the first record in Orders, find the first
   ' matching record in OrderDetails. If no match,
   ' redimension the arrary of order IDs and
   ' add the order ID to the array.

   rstOrderDetails.FindFirst "OrderID = " _
      & rstOrders![OrderID]
   If rstOrderDetails.NoMatch Then
      ReDim Preserve lngOrderID(1 To intIndex)
      lngOrderID(intIndex) = rstOrders![OrderID]
   End If
```

```
    ' We already checked the first record, so we can
    ' find the next record satisfying the criteria
    ' using the FindNext method.
    intIndex = 0

    Do Until rstOrders.EOF
       rstOrderDetails.FindNext _
          "OrderID = " & rstOrders![OrderID]
       If rstOrderDetails.NoMatch Then
          intIndex = intIndex + 1
          ReDim Preserve lngOrderID(1 To intIndex)
          lngOrderID(intIndex) = rstOrders![OrderID]
       End If
       rstOrders.MoveNext
    Loop

ErrorHandler:
Select Case Err
    Case 0
       Exit Function
    Case Else
       MsgBox "Error " & Err & ": " & Error, _
          vbOKOnly, "ERROR"
       Exit Function
End Select

End Function
```

To locate the matching records in reverse order, find the last occurrence with the FindLast method and then use the FindPrevious method.

DAO sets the NoMatch property to True whenever a Find method fails and the current record position is undefined. There may be a current record, but you have no way to tell which one is the current record. If you want to return to the previous current record following a failed Find method, use a bookmark (as described in the following section).

NoMatch is False whenever the operation succeeds. In this case, the current record position is the record found by one of the Find methods.

If you need to frequently search records in a dynaset, you may find it easier to create a temporary indexed table and use the Seek method instead.

Marking Record Position with Bookmarks

A *bookmark* is a system-generated Byte array that uniquely identifies each record. The Bookmark property of a Recordset object changes each time you move to a new record. To identify a record, move to that record and then assign the value of the Bookmark property to a variable of type Variant. To return to the record, set the Bookmark property to the value of the variable.

The following code uses a bookmark to save the current record position and then quickly returns to that record position if a Find or Seek method fails. A bookmark is useful if the method fails, leaving the current record position undefined.

```
Sub UseBookmark()
Dim dbs As Database, rstCustomers As Recordset
Dim varOrigin As Variant

Set dbs = OpenDatabase("Northwind.mdb")
Set rstCustomers = dbs.OpenRecordset _
   ("Customers", dbOpenTable)

rstCustomers.Index = "CompanyName"

' ...Move to other records...

varOrigin = rstCustomers.Bookmark
rstCustomers.Seek ">=", "Z"

If rstCustomers.NoMatch Then
   MsgBox "Can't find a company name" & _
      " starting with 'Z'."
   rstCustomers.Bookmark = varOrigin
End If

rstCustomers.Close
dbs.Close
End Sub
```

This example saves the Bookmark property and then resets the Bookmark property to its previous value when the Seek method fails. This makes the previously current record current again.

The LastModified property of the Recordset object provides a good illustration of how to use a bookmark. The LastModified property returns the bookmark of the last record in the recordset to be added or modified. To use it, set the Bookmark property equal to the LastModified property as follows:

```
rstCustomers.Bookmark = rstCustomers.LastModified
```

This moves the current record position to the last record that was added or modified. This is particularly useful when adding new records, because the current record after adding a new record is the record you were on *before* you added the record. With LastModified, you can move to the newly added record if that's what your application expects.

For More Information For more information, see "Adding a New Record" later in this chapter.

Bookmark Scope

When you close a recordset, any bookmarks you saved become invalid. You can't use a bookmark from one recordset in another recordset, even if both Recordset objects are based on the same underlying table or query. However, you can use a bookmark on the duplicate (clone) of a recordset, as shown in the following code:

```
Dim dbs As Database
Dim rstOriginal As Recordset, rstDuplicate As Recordset
Dim strPlaceholder As String

Set dbs = OpenDatabase("Northwind.mdb")
' Create the first recordset.
Set rstOriginal = dbs.OpenRecordset _
   ("Orders", dbOpenDynaset)

' Save the current record position.
strPlaceholder = rstOriginal.Bookmark

' Create a duplicate recordset.
Set rstDuplicate = rstOriginal.Clone()

' Go to same record.
rstDuplicate.Bookmark = strPlaceholder
```

Why Use Bookmarks Instead of Record Numbers?

If you have used another database or programming environment, you may be accustomed to referring to record numbers. For example, you may have written code that opens a text file and thereafter refers to specific records by their relative position in the file. The first record in the file would be record 1, the second would be record 2, and so on.

In Microsoft Jet databases, your view of records (a recordset) is usually a subset of the records in one or more tables. Because the actual number of records in a recordset can change at any time, especially in a multiuser environment, there's no absolute record number you can always use to refer to a particular record. The AbsolutePosition property isn't the same as a record number, because this property changes if a lower-numbered record is deleted.

Furthermore, records returned in a Recordset object appear in no particular order, unless the recordset was created with a query that includes an ORDER BY clause, or is a table-type recordset with an Index. Record numbers are usually meaningless in a Recordset object.

Instead of record numbers, DAO provides bookmarks to uniquely identify a particular record. A given record retains its unique bookmark for the life of the recordset.

Which Recordset Objects Support Bookmarks?

Dynasets based on certain linked tables, such as Paradox tables that have no primary key, don't support bookmarks, nor do forward-only-type Recordset objects.

You can check the value of the Bookmarkable property to determine whether a given Recordset object supports bookmarks, as in the following example:

```
If rstLinkedTable.Bookmarkable Then
   MsgBox "The underlying table supports bookmarks."
Else
   MsgBox "The underlying table doesn't " _
      & "support bookmarks."
End If
```

Important If you try to use bookmarks on a recordset that doesn't support bookmarks, a run-time error occurs.

Changing Data

After you've created a table- or dynaset-type Recordset object, you can change, delete, or add new records. You can't change, delete, or add records to a Microsoft Jet snapshot-type or forward-only-type Recordset object. However, you can change snapshot data in some ODBCDirect data sources.

This section presents the methods and procedures for changing data in table- and dynaset-type Recordset objects.

Using Parameter Queries

A *parameter query* is a query that when run displays a dialog box that prompts the user for information, such as criteria for retrieving records or a value to insert in a field. You can use stored parameter queries to accomplish most of the database maintenance tasks described in the rest of this chapter.

With Microsoft Jet, a user or another program can provide parameters you can use with your stored queries and Recordset objects. First, create a stored query, specifying which parameters the end user needs to provide. When you open a recordset against one of these queries, the application opens a dialog box that prompts the user to enter a value, such as the criteria for a WHERE clause.

The following function takes two strings representing dates and creates a parameter query that returns all records in the Orders table whose order date is between the two dates. It returns all Order IDs found in the query's recordset in an array.

```
Public Function OrdersFromTo(strDateFrom As String, _
   strDateTo As String, strDB As String, _   lngOrderIDs() As Long)

Dim dbs As Database
Dim rstOrders As Recordset
Dim qdf As QueryDef
Dim strSQL As String
Dim i As Integer

On Error GoTo ErrorHandler

Set dbs = OpenDatabase(strDB)
strSQL = "PARAMETERS [DateFrom] DateTime, " _
   & "[DateTo] DateTime; "
strSQL = strSQL & "SELECT * FROM Orders WHERE " _
   & "OrderDate BETWEEN "
strSQL = strSQL & "[DateFrom] AND [DateTo];"
```

```
' Create an unstored parameter query.
Set qdf = dbs.CreateQueryDef("", strSQL)

' Set the query parameters
qdf.Parameters("DateFrom") = strDateFrom
qdf.Parameters("DateTo") = strDateTo

' Open a forward-only snapshot on the query
Set rstOrders = qdf.OpenRecordset _
   (dbOpenForwardOnly)

' Load all the Order IDs in the query into an
' array that the caller of the function can use.

i = 1
While rstOrders.EOF = False
   ReDim lngOrderIDs(1 To i)
   lngOrderIDs(i) = rstOrders!OrderID
   i = i + 1
   rstOrders.MoveNext
Wend

ErrorHandler:
Select Case Err
   Case 0
      Exit Function
   Case Else
      MsgBox "Error " & Err & ": " & _
         Error, vbOKOnly, "ERROR"
   Exit Function
End Select

End Function
```

Making Bulk Changes

An update or delete query can make changes more efficiently than code that loops through records. The following code creates a QueryDef object to update the Employees table and then runs the query:

```
Dim dbs As Database, qdfChangeTitles As QueryDef

Set dbs = OpenDatabase("Northwind.mdb")
Set qdfChangeTitles = dbs.CreateQueryDef("")

qdfChangeTitles.SQL = "UPDATE Employees " _
   & "WHERE Title = 'Sales Representative';"
```

You can replace the entire SQL string in this example with a stored parameter query, in which case the program would prompt the user for parameter values. The following example shows how the previous example might be rewritten as a stored parameter query:

```
Dim dbs As Database, qdfChangeTitles As QueryDef
Dim strSQLUpdate As String, strOld As String
Dim strNew As String

Set dbs = OpenDatabase("Northwind.mdb")

strSQLUpdate = "PARAMETERS [Old Title] Text, " _
   & "[New Title] Text; " _
   & "UPDATE Employees " _
   & "SET Title = [New Title] " _
   & "WHERE Title = [Old Title];"

' Create the QueryDef object.
Set qdfChangeTitles = dbs.CreateQueryDef _
   ("", strSQLUpdate)

' Prompt for old title.
strOld = InputBox("Enter old job title")

' Prompt for new title.
strNew = InputBox("Enter new job title")

' Set parameters.
qdfChangeTitles.Parameters("Old Title") = strOld
qdfChangeTitles.Parameters("New Title") = strNew
```

A delete query is more efficient than code that loops through records looking for records to delete, especially with Microsoft Jet 3.0 or later databases.

Modifying an Existing Record

You can modify records in a table- or dynaset-type Recordset object by using the Edit and Update methods.

▶ To change an existing record in a table- or dynaset-type Recordset

1. Go to the record that you want to change.
2. Use the Edit method to prepare the current record for editing.
3. Make the necessary changes to the record.
4. Use the Update method to save the changes to the current record.

The following code changes the job titles for all sales representatives in a table called Employees:

```
Sub ChangeTitle()
Dim dbs As Database, rstEmployees As Recordset
Set dbs = OpenDatabase("Northwind.mdb")
Set rstEmployees = dbs.OpenRecordset("Employees")

rstEmployees.MoveFirst
Do Until rstEmployees.EOF
   If rstEmployees!Title = "Sales Representative" Then
      rstEmployees.Edit
      rstEmployees.Update
   End If
   rstEmployees.MoveNext
Loop

rstEmployees.Close
dbs.Close
End Sub
```

Important If you don't use the Edit method before you try to change a value in the current record, a run-time error occurs. If you edit the current record and then move to another record or close the recordset without first using the Update method, your changes are lost without warning. For example, omitting the Update method from the preceding example results in no changes being made to the Employees table.

You can also terminate the Edit method and any pending transactions without saving changes by using the CancelUpdate method. While you can terminate the Edit method by moving off the current record, this isn't practical when the current record is the first or last record in the recordset, or is a new record. It's generally simpler to use the CancelUpdate method.

For More Information For more information about consistent and inconsistent updates, see Chapter 5, "Writing SQL Queries."

Inconsistent Updates

Dynaset-type Recordset objects can be based on a multiple-table query containing the tables with a one-to-many relationship. For example, suppose you want to create a multiple-table query that combines fields from the Orders and Order Details tables. Generally speaking, you can't change values in the Orders table because it's on the "one" side of the relationship. Depending on your application, however, you may want to be able to make changes to the Orders table. To make it possible to freely change the values on the "one" side of a one-to-many relationship, use the dbInconsistent constant of the OpenRecordset method to create an *inconsistent* dynaset.

```
Set rstTotalSales = dbs.OpenRecordset
   ("Sales Totals" ,, dbInconsistent)
```

When you update an inconsistent dynaset, you can easily destroy the relational integrity of the data in the dynaset. You must take care to understand how the data is related across the one-to-many relationship and to update the values on both sides in a way that preserves data integrity.

The dbInconsistent constant is available only for Microsoft Jet dynaset-type Recordset objects. It's ignored for Microsoft Jet table-, snapshot-, and forward-only-type Recordset objects, but no compile or run-time error is returned if dbInconsistent is used with those types of Recordset objects. Dynaset-type and snapshot-type Recordset objects in ODBCDirect data sources support dbInconsistent.

Even with an inconsistent recordset, some fields may not be updatable. For example, you can't change the value of an AutoNumber field, and a recordset based on certain linked tables may not be updatable.

Deleting an Existing Record

You can delete an existing record in a table- or dynaset-type recordset by using the Delete method. You can't delete records from a snapshot-type recordset.

The following function deletes duplicate records in the Shippers table. Duplicate values are determined by the shipper's name.

```
Function DeleteDuplicateShippers() As Integer
   Dim rstShippers As Recordset
   Dim strQuery As String
   Dim dbs As Database
   Dim strName As String

   On Error GoTo ErrorHandler

   strQuery = "SELECT * FROM Shippers" _
      & "ORDER BY CompanyName;"
   Set dbs = OpenDatabase("Northwind.mdb")
   Set rstShippers = dbs.OpenRecordset _
      (strQuery, dbOpenDynaset)

   ' If no records in Shippers table, exit.
   If rstShippers.EOF Then Exit Function

   strName = rstShippers![CompanyName]
   rstShippers.MoveNext
```

```
    Do Until rstShippers.EOF
       If rstShippers![CompanyName] = strName Then
          rstShippers.Delete
       Else
          strName = rstShippers![CompanyName]
       End If
       rstShippers.MoveNext
    Loop

ErrorHandler:
Select Case Err
    Case 0
       DeleteDuplicateShippers = SUCCESS
       Exit Function
    Case Else
       MsgBox "Error " & Err & ": " & _
          Error, vbOKOnly, "ERROR"
       DeleteDuplicateShippers = FAILED
       Exit Function
End Select

End Function
```

When you use the Delete method, Microsoft Jet immediately deletes the current record without any warning or prompting. Deleting a record doesn't automatically cause the next record to become the current record; to move to the next record, you must use the MoveNext method. However, keep in mind that after you've moved off the deleted record, you cannot move back to it.

If you try to access a record after deleting it on a table-type recordset, you'll get error 3167, "Record is deleted." On a dynaset, you'll get error 3021, "No current record."

If you have a recordset clone positioned at the deleted record and you try to read its value, you'll get error 3167 regardless of the type of recordset. Trying to use a bookmark to move to a deleted record will also result in error 3167.

Adding a New Record

You can add a record to a table- or dynaset-type recordset by using the AddNew and Update methods.

▶ To add a new record to a table- or dynaset-type Recordset

1. Use the AddNew method to prepare a new record for editing.
2. Assign values to each of the record's fields.
3. Use the Update method to save the new record.

The following code adds a new record to a table called Shippers.

```
Dim dbs As Database, rstShippers As Recordset

Set dbs = OpenDatabase("Northwind.mdb")
Set rstShippers = dbs.OpenRecordset("Shippers")

rstShippers.AddNew
rstShippers!CompanyName = "Global Parcel Service"

   ' ...Set remaining fields...

rstShippers.Update
rstShippers.Close
dbs.Close
```

When you use the AddNew method, Microsoft Jet prepares a new, blank record and makes it the current record. When you use the Update method to save the new record, the record that was current before you used the AddNew method becomes the current record again.

The new record's position in the recordset depends on whether you added the record to a dynaset- or table-type recordset. If you add a record to a dynaset-type recordset, the new record appears at the end of the recordset, no matter how the recordset is sorted. To force the new record to appear in its properly sorted position, you can use the Requery method or recreate the Recordset object.

If you add a record to a table-type recordset, the record appears positioned according to the current index, or at the end of the table if there is no current index. Because Microsoft Jet 3.0 or later allows multiple users to create new records in a table simultaneously, your record may not appear right at the end of the recordset as it did in previous versions of Microsoft Jet. Be sure to use the LastModified property rather than MoveLast to move to the record you just added.

Important If you use the AddNew method to add a new record, and then move to another record or close the recordset without first using the Update method, your changes are lost without warning. For example, omitting the Update method from the preceding example results in no changes being made to the Shippers table.

Caching ODBC Data with a Recordset

You can use the dynaset-type recordset to create a local cache for Microsoft Jet-connected ODBC data sources. This lets you retrieve records in batches instead of one at a time as each record is requested, and makes much better use of your server connection, thus improving performance.

The CacheSize and CacheStart properties establish the size and starting offset (expressed as a bookmark) for the cache. For example, set CacheSize to 100 records. Using the FillCache method, you can retrieve sufficient records to fill the cache.

For More Information For more information about caching ODBC data, see Chapter 9, "Developing Client/Server Applications."

Tracking Recordset Changes

You may need to determine when the underlying TableDef object of a table-type recordset was created, or the last time it was modified. The DateCreated and LastUpdated properties, respectively, give you this information. Both properties return the date stamp applied to the table by the computer storing the table at the time it was stamped. These properties are only updated when the table's design changes; they aren't affected by changes to records in the table.

Microsoft Jet Transactions

A *transaction* is a set of operations bundled together and treated as a single unit of work. The work in a transaction must be completed as a whole; if any part of the transaction fails, the entire transaction fails. Transactions offer the developer the ability to enforce data integrity. With multiple database operations bundled into a single unit that must succeed or fail as a whole, the database can't reach an inconsistent state. Transactions are common to most database management systems.

The most common example of transaction processing involves a bank's automated teller machine (ATM). The processes of dispensing cash and then debiting the user's account are considered a logical unit of work and are wrapped in a transaction: The cash isn't dispensed unless the system is also able to debit the account. By using a transaction, the entire operation either succeeds or fails. This maintains the consistent state of the ATM database.

Note The behavior of transactions with Microsoft Jet databases differs from the behavior of ODBC data sources, such as Microsoft SQL Server. For example, if a database is connected to a file server, and the file server stops before a transaction has had time to commit its changes, then your database could be left in an inconsistent state. If you require true transaction support with respect to durability, you should investigate the use of a client/server architecture. For more information about client/server architecture, see Chapter 9, "Developing Client/Server Applications."

Using Transactions in Your Applications

Microsoft Jet supports transactions through the DAO BeginTrans, CommitTrans, and Rollback methods of the Workspace object. The basic syntax is shown in the following table.

Method	Operation
Workspace.BeginTrans	Begins the transaction.
Workspace.CommitTrans	Posts the transaction, writing its updates to the permanent database objects.
Workspace.Rollback	Cancels the transaction.

The following code changes the job title of all sales representatives in the Employees table of the Northwind.mdb database. After the BeginTrans method starts a transaction that isolates all of the changes made to the Employees table, the CommitTrans method saves the changes. Notice that you can use the Rollback method to undo changes that you saved with the Update method:

```
Sub ChangeTitle()

   Dim dbsEmployees As Database
   Dim rstEmp As Recordset
   Dim wspCurrent As Workspace

   Set wspCurrent = DBEngine.Workspaces(0)
   Set dbsEmployees = OpenDatabase("Northwind.mdb")
   Set rstEmp = dbsEmployees.OpenRecordset _
      ("Employees", dbOpenTable)

   wspCurrent.BeginTrans
   Do Until rstEmp.EOF

      If rstEmp!Title = "Sales Representative" Then
         rstEmp.Edit
         rstEmp!Title = "Sales Associate"
         rstEmp.Update
      End If

      rstEmp.MoveNext
   Loop
```

```
    If MsgBox("Save all changes?", _
       vbQuestion + vbYesNo) = vbYes Then
       wspCurrent.CommitTrans
    Else
       wspCurrent.Rollback
    End If

    rstEmp.Close
    dbsEmployees.Close

End Sub
```

When you use transactions, all databases and Recordset objects in the specified Workspace object are affected — transactions are global to the workspace, not to a specific database or recordset. If you perform operations on more than one database or within a workspace transaction, the Commit and Rollback methods affect all the objects changed within that workspace during the transaction.

Note You can also use the BeginTrans, Rollback, and CommitTrans methods with the DBEngine object. In this case, the transaction is applied to the default workspace, which is `DBEngine.Workspaces(0)`.

Managing Transactions

Microsoft Jet uses sophisticated algorithms to enhance transaction performance, reliability, and usability. This section discusses topics related to how the engine manages transactions.

The Temporary Database

Almost all database transaction systems store intermediate changes in a temporary log file instead of writing them directly to the database. Similarly, Microsoft Jet buffers all transaction activity to a temporary database. When the transaction is committed, the contents of the temporary database are merged into the real database. If you issue a Rollback method to cancel the transaction, the engine frees the pages in the temporary database.

Microsoft Jet doesn't create the temporary database until it has to. It uses whatever cache memory is available to store changes to data. After the cache is exhausted, the engine creates the temporary database and starts to write changes there.

Microsoft Jet creates the temporary directory specified by the TEMP environment variable of the workstation. If the available disk space for the temporary database is exhausted during a transaction, a trappable run-time error occurs. Your code should check for this error (number 2004) and react accordingly. If you attempt to commit the transaction after this error occurs, the engine will commit an indeterminate number of changes, possibly leaving the database in an inconsistent state. You should usually roll back the transaction when this error occurs to ensure a consistent database state.

Although the temporary database is a Microsoft Jet database, it's used internally by the engine only. It cannot be opened from other applications. After a transaction is complete, the engine frees the pages in the temporary database.

Note Only the amount of physical space on the disk drive holding the temporary database limits transaction size. That is, Microsoft Jet can store a quantity of transaction data as large as the amount of free space on your hard disk.

Nesting Transactions

You can have up to five levels of transactions active at any one time by nesting combinations of BeginTrans and CommitTrans or Rollback. If you nest transactions, you must make sure that you commit or roll back the current transaction before trying to commit or roll back a transaction at a higher level of nesting.

If you want to have transactions with overlapping, nonnested scopes, you can open additional Workspace objects and manage other transactions within those new workspaces.

When a Transaction is Rolled Back by the Engine

If you close a database variable, any uncommitted transactions that were within the scope of that database variable are automatically rolled back. Microsoft Jet never automatically commits any transactions you have started. This behavior is also true of the Workspace object. If you close a Workspace object, any transactions within the scope of the workspace are rolled back. You should be aware of this behavior when you write your code. Never assume that the engine is going to commit your transaction for you.

Transactions on External Data Sources

Transactions aren't supported on external data sources other than Microsoft Jet, with the exception of ODBC data. For example, if your database has linked FoxPro or dBASE tables, the database engine ignores any transactions on those objects. This means that the transaction will not fail or generate a run-time error, but it won't actually do anything, either.

Microsoft Jet opens Microsoft Jet 2.*x* databases as external ISAM databases. However, unlike the other nonnative external data sources, the engine does support transactions on Microsoft Jet 2.*x* databases.

You can check the value of the Transactions property on Database and Recordset objects to determine whether or not the object supports transactions. A value of True indicates that the object supports transactions, and a value of False indicates that the object doesn't support transactions.

Transactions and Performance

In versions 1.*x* and 2.*x* of Microsoft Jet, it was generally recommended that you use transactions as a performance enhancement. In almost every case, wrapping DAO updates in a transaction was much faster because the transaction forced the engine to buffer changes in the cache rather than write them directly to disk. Various performance enhancements in Microsoft Jet 3.5 remove the need to use transactions for performance reasons.

Microsoft Jet 3.5 provides automatic, internal transactions for all DAO add, update, and delete operations. In most situations, this automatic support provides your application with the best possible performance. However, there may be situations in which you want to fine-tune the engine's internal transaction behavior. You can create and modify various settings in the Windows Registry so that the engine can handle your data better.

Synchronous and Asynchronous Processing

Microsoft Jet has two modes of operation for transaction processing: synchronous and asynchronous. Synchronous processing is the only mode available in version 2.*x* of the engine.

Synchronous processing prevents anything else from occurring until the current operation is completed. This is the model that Microsoft Jet 2.*x* follows.

Asynchronous processing allows the engine to queue up a series of changes to the database. For example, several Update methods could be grouped together. At some point, these updates are written to disk and the engine begins grouping new changes. The updates occur when one of the following happens.

- A specified period of time passes from the first stored change (controlled by the ExclusiveAsyncDelay and SharedAsyncDelay Windows Registry settings).
- Internal memory set aside for this (as defined by the MaxBufferSize Windows Registry setting) is exceeded.

To better understand these modes, consider the following DAO code:

```
Set rst = dbs.OpenRecordset("SELECT * FROM Customers")
While Not rst.EOF
   rst.Edit
   '...Make edits here...
   rst.Update
   rst.MoveNext
Wend
```

With synchronous processing, Microsoft Jet doesn't continue the loop until `rst.Update` completes successfully.

With asynchronous processing, Microsoft Jet immediately processes the `rst.Update` method, and because it uses a thread to do the work, it continues on another thread to process the `rst.MoveNext` operation.

Microsoft Jet places temporary files according to paths specified by the TEMP environment variable. The behavior is the same for Windows 95 and Windows NT.

Extracting Data from a Record

After you've located a particular record or records, you may want to extract data to use in your application instead of modifying the underlying source table.

Copying a Single Field

You can copy a single field of a record to a variable of the appropriate data type. The following code extracts three fields from each record in a Recordset object.

```
Dim dbs As Database, rstEmployees As Recordset
Dim strFirstName As String, strLastName As String
Dim strTitle As String

Set dbs = OpenDatabase("Northwind.mdb")
Set rstEmployees = dbs.OpenRecordset("Employees")

rstEmployees.MoveFirst
strFirstName = rstEmployees!FirstName
strLastName = rstEmployees!LastName
strTitle = rstEmployees!Title
```

Copying Entire Records to an Array

To copy one or more entire records, you can create a two-dimensional array and copy records one at a time. You increment the first subscript for each field and the second subscript for each record.

A fast way to do this is with the GetRows method. The GetRows method returns a two-dimensional array. The first subscript identifies the field and the second identifies the row number, as follows:

```
varRecords(intField, intRecord)
```

The following code uses an SQL statement to retrieve three fields from a table called Employees into a Recordset object. It then uses the GetRows method to retrieve the first three records of the recordset, and it stores the selected records in a two-dimensional array. Each record is then printed, one field at a time, by using the two array indexes to select specific fields and records.

To clearly illustrate how the array indexes are used, the following code uses a separate statement to identify and print each field of each record. In practice, it would be more

reliable to use two loops, one nested in the other, and to provide integer variables for the indexes that step through both dimensions of the array.

```
Sub GetRowsTest()

   Dim dbs As Database
   Dim rstEmployees As Recordset
   Dim varRecords As Variant
   Dim intNumReturned As Integer
   Dim intNumColumns As Integer
   Dim intColumn As Integer, intRow As Integer

   Set dbs = OpenDatabase("Northwind.mdb")
   Set rstEmployees = dbs.OpenRecordset _
      ("SELECT FirstName, LastName, Title " _
      & "FROM Employees", dbOpenSnapshot)

   varRecords = rstEmployees.GetRows(3)
   intNumReturned = UBound(varRecords, 2) + 1
   intNumColumns = UBound(varRecords, 1) + 1

   For intRow = 0 To intNumReturned - 1
      For intColumn = 0 To intNumColumns - 1
         Debug.Print varRecords _
                  (intColumn, intRow)
      Next intColumn
   Next intRow

   rstEmployees.Close
   dbs.Close

End Sub
```

You can use subsequent calls to the GetRows method if more records are available. Because the GetRows method fills the array as soon as you call it, this approach is much faster than using assignment statements to copy one field at a time.

Also notice that you don't have to declare the Variant as an array, because this is done automatically when the GetRows method returns records. This enables you to use fixed-length array dimensions without knowing how many records or fields will be returned, instead of using variable-length dimensions that take up more memory.

If you're trying to retrieve all the rows by using multiple GetRows calls, use the EOF property to be sure that you're at the end of the recordset. GetRows may return fewer records than what you request. For example, if the fifth record cannot be retrieved in a group of ten records that you're trying to retrieve, GetRows returns four records and leaves the current record position on the record that caused a problem—and doesn't generate a run-time error. This situation may occur if a record in a dynaset was deleted by another user. If fewer records were returned than the number requested and you're not at the end of the file, you will need to read each field in the current record to determine what error GetRows encountered.

Because GetRows always returns all the fields in the recordset, you may want to create a query that returns just the fields that you need. This is especially important for Memo and Long Binary (OLE Object) fields.

Using Field Objects

The default collection of a Recordset object is its Fields collection. This collection includes a single Field object corresponding to each field (or column) in the recordset. Each Field object has a set of properties uniquely identifying it, such as the field name and data type, as well as the value of the field in the current record.

Referencing Field Objects

You can identify a Field object by its Name property, which corresponds to the column name in the table from which the data in the field was retrieved. The Fields collection is the default collection of a Recordset object. Therefore, to refer to the LastName field in a recordset called `rstEmployees`, you can use any of the following syntax.

```
rstEmployees.Fields("LastName")
rstEmployees!LastName
rstEmployees![LastName]
```

When using the ! operator, you must include brackets around a field name when it contains spaces. For example, the statement

```
strEmp = rstEmployees!Last Name
```

will not compile, but the statement

```
strEmp = rstEmployees![Last Name]
```

will compile with no problems.

You can also identify each Field object by its index within the Fields collection:

```
rstEmployees.Fields(0)
```

You use the index to walk through the collection in a loop, replacing the index with a variable that is incremented with each pass through the loop. Object numbers in the collection are zero-based, so the first Field is number 0, the second is 1, and so on. The underlying table determines the field order. Fields are usually numbered in the order retrieved when the recordset is opened. One drawback to this approach is that you can't be certain which field will be referenced, because the underlying table structure may change, as fields are added or deleted.

To help you determine the order of fields in a Fields collection, the Field object supports the OrdinalPosition property, which you can use to get or set a field's position relative to other fields in the collection. You can set the OrdinalPosition property to any positive integer to change the field order when data is displayed in a form, or copied to an array or a Microsoft Excel worksheet.

When writing code that references fields within a loop, it's more efficient to reference Field objects rather than to refer to fields by their name. The following code is a more efficient way of writing the ChangeTitle procedure discussed earlier in this chapter. Instead of referring to the Title field as `rstEmp!Title`, it refers to the field by its object variable, which doesn't require your code to look up the field in the Fields collection every time it's referenced.

```
Sub ChangeTitle()

   Dim dbsSales As Database
   Dim rstEmp As Recordset, fldTitle As Field
   Dim wspCurrent As Workspace

   Set wspCurrent = DBEngine.Workspaces(0)
   Set dbsSales = OpenDatabase("Northwind.mdb")
   Set rstEmp = dbsSales.OpenRecordset _
      ("Employees", dbOpenTable)

   Set fldTitle = rstEmp.Fields("Title")
   wspCurrent.BeginTrans
   Do Until rstEmp.EOF

      If fldTitle = "Sales Representative" Then
         rstEmp.Edit
         fldTitle = "Sales Associate"
         rstEmp.Update
      End If

      rstEmp.MoveNext
   Loop

   If MsgBox("Save all changes?", _
      vbQuestion + vbYesNo) = vbYes Then
      wspCurrent.CommitTrans
   Else
      wspCurrent.Rollback
   End If

   rstEmp.Close
   dbsSales.Close

End Sub
```

Field Data Types

A Field object has a Type property that can be set to one of the following 21 DAO data types.

Type property setting	Data type
dbBigInt	Big Integer
dbBinary	Binary
dbBoolean	Boolean
dbByte	Byte
dbChar	Char
dbCurrency	Currency
dbDate	Date/Time
dbDecimal	Decimal
dbDouble	Double
dbFloat	Float
dbGUID	GUID
dbInteger	Integer
dbLong	Long
dbLongBinary	Long Binary (OLE Object)
dbMemo	Memo
dbNumeric	Numeric
dbSingle	Single
dbText	Text
dbTime	Time
dbTimeStamp	Time Stamp
dbVarBinary	VarBinary

For a Field object in a Recordset object, the Type property is read-only. However, you must be aware of the field type when copying data to or from a field in code or a "Type mismatch" error may occur. For example, you cannot copy Text data to an Integer field.

The underlying table determines the Type property of a Field in a recordset. If you create the table and its fields using DAO data definition language (DDL) statements, you can easily determine the data type of the table's fields. However, if you're retrieving data from tables created with Microsoft Jet SQL DDL statements, the data type you specify when you create the table may not match one of the 21 DAO data types.

If you're accessing external data through an installable ISAM driver, the data types within external tables may be different from those defined within Microsoft Jet. The ISAM driver for the external data source converts external data types into their closest equivalent DAO data type. For more information about data type conversion in external data sources, see Chapter 7, "Accessing External Data."

Note A Field object's data type is read/write before the field is appended to a table's Fields collection, and read-only after it's appended.

The GUID Data Type

The GUID type is used to store a globally unique identifier, a unique string of digits that identifies an object class, Microsoft SQL Server remote procedure calls, and other entities that need a unique reference identification. The GUID type is also used in Microsoft Jet database applications to identify a replica, such as the Database object's ReplicaID property.

The Text Data Type

For Field objects declared as type dbText, you must set the Size property, which indicates the length of the longest string that can be stored in the field. All other types of Field objects have their Size property set automatically.

The Currency Data Type

If you need to store monetary values, use fields of type dbCurrency. Don't use any of the number data types (such as dbSingle) for currency values, because numbers to the right of the decimal may be rounded during calculations. The dbCurrency data type always maintains a fixed number of digits to the right of the decimal.

The Long Data Type

In some tables, you'll want to store a series of sequential numbers to uniquely identify records. For example, you may want to start customer order records at order number 1 and begin counting upward.

The Microsoft Jet database engine can automatically insert unique numbers in a field, so that your application doesn't need to generate unique identifiers for a primary key field. To take advantage of this capability, define a field with the dbLong data type and set dbAutoIncrement in the Field object's Attributes property. Auto-incrementing fields start at 1 and increment sequentially. Fields of this type are also referred to as *AutoNumber fields*.

If you want to establish a primary key/foreign key relationship between two tables using an AutoNumber field, make sure that the foreign key field is also defined as dbLong. You can also set dbRandomIncrField in a Field object's Attributes property to tell Microsoft Jet to automatically fill a field of type dbLong with random numbers.

Long Binary and Memo Data Types

Long Binary (OLE Object) and Memo fields are collectively referred to as "large value" fields because they are typically much larger than fields of other data types. Large value fields of type dbLongBinary consist of binary data up to 1.2 gigabytes in size. You can use Long Binary fields to store pictures, files, or other raw binary data. You can use Memo fields to store lengthy text and numbers, such as comments or explanations. The maximum size of the database limits the size of a Memo field.

Records within a Recordset object must fit on the 2K pages supported by Microsoft Jet. Each Field object you include in the table definition counts toward this 2K total, including Long Binary and Memo fields. However, the amount stored for Long Binary and Memo fields is only 14 bytes per non-Null field, and only 1 byte for Null fields. The 14 bytes store a pointer to the actual data for these fields, which is located on additional 2K pages. The amount of data committed to each text field isn't set until you actually store data in the field. You can define more text fields than space could allow, but no more than about 2K of actual data can be stored in a record. For example, you can define 15 250-byte text fields in a record, but the total number of characters stored must be less than about 2K.

When you query tables containing large value fields, don't include those fields in the field list unless you need them, because returning large value fields takes time. Also be aware that you can't index large value fields.

A snapshot-type or forward-only –type recordset opened against large value fields in an .mdb file doesn't actually contain that data. Instead, the snapshot maintains references to the data in the original tables, the same way a dynaset references all data.

Handling Large Value Data

Sometimes you'll need to read or copy data from a large value field where you don't have sufficient memory to copy the entire field in a single statement. You instead have to break up the data into smaller units, or "chunks," that will fit available memory. The FieldSize property tells you how large the field is, measured in bytes. Then you can use the GetChunk method to copy a specific number of bytes to a buffer, and use the AppendChunk method to copy the buffer to the final location. You then continue using GetChunk and AppendChunk until the entire field is copied.

For More Information For more information about DAO field data types, see "Type Property" In the *Language Reference.*

Reading and Writing Data

When you read and write data to a field, you're actually reading or setting the Value property of a Field object. The Value property is the default property of a Field. Therefore, to set the Value property of the LastName field in the Recordset object called `rstEmployees`, you can use any of the following syntax:

```
rstEmployees!LastName.Value = strName
rstEmployees!LastName = strName
rstEmployees![LastName] = strName
```

Write Permission

You may not be able to modify the underlying tables of a Recordset object, even though it is a dynaset- or table-type recordset, which are usually updatable. Check the Updatable property of the Recordset object to determine whether its data can be changed. If the property is True, you can modify the data in the recordset.

Individual fields within an updatable recordset may not be updatable, and attempting to write to these fields generates a run-time error. To determine whether a field is updatable, check the DataUpdatable property of the corresponding Field object in the Fields collection of the recordset.

The following function takes an SQL SELECT string, creates a dynaset on the SELECT string, and returns True if all fields in the dynaset are updatable. Otherwise, the function returns False.

```
Function UpdatableEx(strQuery As String, StrDB As _ String) As Integer

   Dim dbs As Database, rstDynaset As Recordset
   Dim i As Integer

   On Error GoTo ErrorHandler

   Set dbs = OpenDatabase(StrDB)
   Set rstDynaset = dbs.OpenRecordset _
      (strQuery, dbOpenDynaset)

   ' If the entire dynaset isn't updatable,
   ' return False.

   If rstDynaset.Updatable = False Then
      UpdatableEx = False
   Else
   ' If the dynaset is updatable, determine if
   ' all fields in the dynaset are updatable.
   ' If at least one of the fields is not updatable,
   ' return False.

      For i = 0 To rstDynaset.Fields.Count - 1
         If rstDynaset.Fields(i).DataUpdatable _
            = False Then
            UpdatableEx = False
            Exit For
         End If
      Next i
   End If

ErrorHandler:
Select Case Err
   Case 0
      Exit Function
   Case Else
      MsgBox "Error " & Err & ": " & _
         Error, vbOKOnly, "ERROR"
      Exit Function
End Select

End Function
```

Note If at least one of the fields in a dynaset-type recordset is updatable, then the Updatable property on the recordset object will be True.

Criteria

Any single field can impose a number of criteria on data in that field when records are added or updated. Several properties define these criteria. The AllowZeroLength property on a Text or Memo field indicates whether or not the field will accept a zero-length string. The Required property indicates whether or not some value must be entered in the field, or if it instead can accept a Null. For a Field object in a Recordset object, these properties are read-only; the underlying table determines their state.

Field-Level Data Validation

Validation is the process of determining whether data entered into a field's Value property is within an acceptable range. A Field object on a recordset may have the ValidationRule and ValidationText properties set. You can use the ValidationText for a message that you want to display to the user if the user attempts to enter data that is outside the limits of the ValidationRule property.

Note The ValidationRule and ValidationText properties also exist at the recordset level. These are read-only properties, reflecting the table-level validation scheme established on the base table from which the current record is retrieved.

A Field object in a Recordset object also features the ValidateOnSet property. When ValidateOnSet is True, validation is checked as soon as the field's Value property is set. If it's False (the default), validation is checked only when the completed record is updated. For example, if you're adding data to a record containing a large Memo or Long Binary (OLE Object) field with a ValidationRule, you should determine whether the new data violates the validation rule before attempting to write the data. You should write the data when the field value is set, not when the entire record is written to disk. This way, you don't waste time attempting to write an invalid record to disk.

Tracing the Origin of Dynaset Fields

A dynaset-type recordset can include records from more than one source table. Also, a single record can include fields joined from different tables. Sometimes it's useful to know the table from which a field originated. The SourceTable property of a Field object returns the name of the base table from which the field's current data was retrieved.

Within a query, a field can be renamed for display purposes. For example, in an SQL SELECT query, the AS operator in the select field list can create an alias for one of the returned fields. In a recordset based on an SQL query, a field that has been aliased is represented by a Field object whose Name property reflects the alias, not the original field name. To find out the original field name, check the Field object's SourceField property.

CHAPTER 4

Incorporating Visual Basic Controls

Visual controls are one of the greatest advantages in Visual Basic, allowing you to quickly and easily create sophisticated, visually appealing user interfaces with a minimum of effort. With database applications, the data control and bound controls make it possible to create visual interfaces to the database with little or no code.

The data control and data-bound controls are presented in Chapter 14, "Accessing Data," in the *Programmer's Guide*. This chapter will briefly focus on enhanced coding techniques you can use with these controls, in addition to the new flexibility provided by the data access objects (DAO) programming interface.

In addition, you'll find information on how you can create your own data-bound controls, and how to use standard (unbound) controls with the data control to indirectly manipulate data extracted from a database.

Using the Data Control with Data Access Objects

All editions of Visual Basic let you use the data control to perform much of the database management for your application. In the Professional and Enterprise editions, however, you also have the ability to create whatever data access objects you need. You can use these data access objects together with the data control to increase your flexibility in working with a database. You can perform the following tasks with the Professional and Enterpise editions:

- Create new Recordset variables based on the original Recordset variable provided by the data control.
- Modify the structure of a database by adding tables, fields, or indexes, or by removing tables and indexes.
- Use other Database and Recordset methods not available in the Learning edition, including the Seek, Find, OpenRecordset, and CreateDatabase methods.
- Declare data object variables and use them as "shorthand" notation for the Database and Recordset properties as well as other data object variables.

Using the Database and Recordset Properties

Once you have opened a database with the data control, you can use the control's Database and Recordset properties just as if they were created by the OpenDatabase method. For example, suppose you have created a data control with the following properties.

Property	Setting
DatabaseName	Biblio.mdb
RecordsetType	1-Dynaset
RecordSource	Publishers

The following code:

```
Dim Db As Database, Rs As Recordset
Set Db = Data1.Database
Set Rs = Data1.Recordset
```

is equivalent to this code:

```
Dim Db As Database, Rs As Recordset
Set Db = _
DBEngine.Workspaces(0).OpenDatabase("Biblio.mdb")
Set Rs = Db.OpenRecordset("Publishers")
```

Because of this, you can use either variety of these variables to perform any of the Database and Recordset methods, examine any of their properties, modify any of the TableDefs collection members, or modify any of the Fields, Indexes, Relations, or QueryDefs collection members directly in code.

Conversely, you can also assign any Recordset object that you create using DAO to any data control, as follows:

```
Dim Db As Database, Rs As Recordset
Set Db = _
DBEngine.Workspaces(0).OpenDatabase("Biblio.mdb")
Set Rs = Db.OpenRecordset("Publishers")
Set Data1.Recordset = Rs
```

When you assign a new Recordset object to the data control, the related properties (DatabaseName, RecordSource, and so on) are automatically changed to reflect the new recordset. You must make sure that the DataField properties of any bound controls attached to the data control are set to valid field names in the new Recordset.

Professional Edition Bound Controls

The Professional and Enterprise editions provide additional data-bound controls, beyond those included in the Learning edition. These controls include:

- ProgressBar control

  The ProgressBar control shows the progress of time-consuming operations by slowly filling in a rectangle from left to right. This can be useful when loading large recordsets or performing complex, time-consuming queries.

- RichTextBox control

  The RichTextBox control displays and edits text in RTF format, allowing for advanced formatting features such as font changes, size and style differences, text justification and so on. This provides a more attractive text display as well as word processor-like features that can be incorporated into your database applications.

- Masked Edit control

  The masked edit control lets you define a "mask," or predefined format, for the type of data that will be entered. This makes it useful for fields such as telephone numbers, currency amounts, and other values where you want to display the number with extraneous characters (such as commas and parentheses), or limit the kinds of information that can be entered (such as only numerical digits or only uppercase characters).

For More Information For general information on bound controls and how they interact with the data control, see Chapter 14, "Accessing Data," in the *Programmer's Guide*. For more information about individual ActiveX controls, see Chapter 7, "Using Visual Basic's Standard Controls," in the *Programmer's Guide* and "Using ActiveX Controls" in the *Component Tools Guide*.

Creating Your Own Bound Controls

One of the most exciting new features of Visual Basic, version 5.0, is the ability to create your own ActiveX controls. This capability is discussed in detail in "Creating ActiveX Components" in the *Component Tools Guide*.

Even more important for database programmers, you can create your own data-bound controls by using the DataBindings collection, an extender property that Visual Basic provides for your control.

▶ Steps to create a bound control

1 Create the control, using labels, text boxes or other standard controls as the element(s) you will make data-bound.

2 Create properties for the elements that will be data-bound, to enable you to change their contents. For example, if you use a text box to bind a "Company" field, you might write the following code:

```
Public Property Get Company() As String
   Company = txtCompany.Text Then
End Property

Public Property Let Company(NewValue As String)
   If CanPropertyChange("Company")
      ' Set the value.
      txtCompany.Text = NewValue
      ' Update the data source.
      PropertyChanged "Company"
   End If
End Property
```

3 On the **Advanced** section of the **Procedure Attributes** dialog box, available on the **Tools** menu, mark the property as data bound.

4 Also on the **Advanced** section, mark the property as bound to the control's DataField.

Adding Multiple Bound Properties to a Control

Like the bound controls supplied with Visual Basic, your data-bound control will have a DataSource property and DataField property. You can create as many bound properties as you wish on your control; however, only one of them can be associated with the DataField in the Procedure Attributes dialog box, as indicated above. To allow values to be set for additional bound properties, you will have to use the DataBindings collection.

For each additional bound property you add to your control, check Show in Databindings Collection at Design Time on the Advanced section of the Procedure Attributes dialog box.

At design time, the DataBindings collection will appear in your control's property page. Clicking on the ellipsis next to this property will open the Data Bindings dialog box, allowing you to set the DataField for each bound property on your control.

You can also set the DataField for additional bound properties at run time, using Visual Basic code. For example, a "Contacts" control with a second data bound field, "Telephone," could be set with the following code:

```
Contacts.DataBindings( _
   "Telephone").DataField = "WorkPhone"
```

(This example assumes there is a "WorkPhone" field in the database to which the control is bound, as specified in the DataSource property.)

For More Information For a complete discussion of creating bound controls, see "Binding a Control to a Data Source" in Chapter 9 of *Creating ActiveX Components*.

Using Unbound Controls

The data control and bound controls give you tremendous flexibility in designing the user interface for your database application. There may be occasions, however, when it makes sense to use unbound controls to display data and make changes to a recordset. It is perfectly possible to do this; however, the database will not automatically save changes based on changes you make in unbound controls, so you must make any updates in code using DAO methods.

An example of using an unbound control with data access objects is creating a list box to display a set of values from a recordset, where you don't want the user's selection to automatically update the recordset as it would with the bound list box control. (Actually, the standard list box does contain limited data-binding capabilities, but it does not accomplish this purpose, so this example uses the list box as an unbound control.) For example, you might want to display the names of all the publishers in the Biblio.mdb sample database. Once the user makes a selection from the list, you can read that value and use it any way you want.

The following example uses the data control to manually fill a list box with publishers' names.

```
Dim Rs As Recordset
Data1.DatabaseName = "Biblio.mdb"
Data1.RecordsetType = 1        ' Dynaset-type Recordset.
Data1.RecordSource = "SELECT Name FROM Publishers"
Data1.Refresh                  ' Open the database.

PubsList.Clear                 ' Clear out the list box.
Set Rs = Data1.Recordset

Do While Rs.EOF = False
   PubsList.AddItem Rs(0)      ' Add the field.
   Rs.MoveNext                 ' Move to the next field.
Loop
```

At run time, this code fragment produces a list of all valid publisher names and adds them to the list box.

Using Bound and Unbound Controls Together

It is also possible to use both bound and unbound controls together. For example, you might want to use the list box in the previous example to allow the user to select a particular publisher, and then have that selection automatically fill a data control with a recordset containing titles published by that publisher. To accomplish this, you could simply read the Text property of the list box (in the Click event procedure) to obtain the publisher's name and use it to build an SQL query for the data control.

Working with Multiple Recordsets

In working with bound and unbound controls, it is often useful to have duplicate versions of the same recordset, which can be independently manipulated. You can use the Clone method to create multiple versions of the same recordset, allowing you to have more than one pointer into a recordset at a time. You can also use bookmarks from one recordset to position a record within a clone of that recordset, so you can easily update the position of one recordset to correspond to the position of its clone.

For example, to clone a recordset object created with a data control, and position two current record pointers within that object, you could write the following code:

```
Sub GetData_Click ()
' Set up variables to reference each Table object.
Dim i As Integer
Dim Rs As Recordset      ' These are Form-level
Dim RsClone As Recordset' variables, so they can be
                         ' used by the other controls.
Dim BookMarks(2000)
' Open the database.
Data1.DatabaseName = "Biblio.mdb"
Data1.RecordSource = "Titles"
Data1.Refresh
Set Rs = Data1.Recordset    ' Address first Recordset.
Set RsClone = Rs.Clone()    ' Clone the Recordset
                            ' object.

' At this point you can reposition the record pointer
' in the Rs and RsClone independently.

List1.Clear ' Each list will be sorted independently
List2.Clear ' based on its contents.
```

```
' Fill the list boxes - each with one of the Dynasets.
i = 0
Do While Not Rs.EOF And i < 2001
   List1.AddItem Rs("Title")
   List2.AddItem RsClone("ISBN")
   BookMarks(i) = Rs.Bookmark
   i = i + 1
   ' Move to the next record in each Recordset.
   Rs.MoveNext
   Rsclone.MoveNext
Loop
' Each of these two Lists has the same information but
' each can be scrolled separately, just like the
' Dynasets used to build them.
End Sub
```

Once the list boxes are filled, the user can scroll through any one of the list boxes and choose one of the items. The following list box Click events set the Bookmark property of the original recordset to the value stored when the list was built, based on the ListIndex property.

```
Sub List2_Click ()
   Rs.Bookmark = BookMarks(List2.ListIndex)
End Sub
Sub List1_Click ()
   Rs.Bookmark = BookMarks(List1.ListIndex)
End Sub
```

This scheme can be used in conjunction with the data control and standard bound controls that display the entire record. Once the user clicks an item in one of the lists and sets the corresponding bookmark, the bound controls are updated with the record that contains the selected value in the list box. For example, if the user selected a title from the first list box, the bound controls would display the entire record for that title. If the user selected an ISBN number from the second list box, the bound controls would display data from the record for that ISBN.

CHAPTER 5

Writing SQL Queries

The Structured Query Language (SQL) is an industry-standard database language used by the Microsoft Jet database engine. SQL is used to create QueryDef objects, as the source argument to the OpenRecordset method, and as the RecordSource property of the data control. It can also be used with the Execute method to directly create and manipulate Jet databases, and to create SQL PassThrough queries to manipulate remote client-server databases.

This chapter will address the basic structure of SQL, and the ways it is used for creating, maintaining, and modifying databases. It will also cover the construction and use of SQL queries to create Recordset objects, and to select, sort, filter, and update data in the underlying database tables. In addition, this chapter will examine ways of optimizing SQL queries for speed and performance. Finally, the differences between Microsoft Jet SQL and ANSI SQL will be presented in detail.

What is SQL?

SQL is a database programming language with origins closely connected to the invention of the relational database by E. F. Codd in the early 1970's. An early forerunner of today's SQL was the *Sequel* language, and for this reason SQL is still commonly pronounced like "sequel" rather than "ess cue ell," although both pronunciations are acceptable.

Modern SQL has evolved into a widely used standard for relational databases, and is defined by the ANSI standard. Most implementations of SQL have minor variations from the defined standard, including the version supported by the Jet database engine. These differences will be addressed later in this chapter, but the overall structure and functionality of the language is very consistent from vendor to vendor. If you have used any implementation of SQL, you will have no difficulty making the transition to the Microsoft Jet version.

SQL vs. Navigation

As described earlier in this manual, the Microsoft Jet database engine provides two separate methods for accomplishing most database tasks:

- A *navigational* model that is based on moving around in the database records.
- A *relational* model that is based on the structured query language (SQL).

The navigational model consists of the properties and methods described in Chapter 2, "Creating and Modifying Databases," and Chapter 3, "Working with Records and Fields." The relational model is described in this chapter.

Programmers who are familiar with file-oriented database systems such as dBASE, FoxPro, and Paradox might feel more comfortable beginning with the navigational methods discussed in the previous chapters mentioned above. In most cases, however, the equivalent SQL methods are more efficient, and they should generally be used where performance considerations are important. In addition, SQL has the advantage of being an industry-standard database interface, so a knowledge of SQL commands allows you to access and manipulate a wide variety of database products from many different vendors.

In practice, you will often find yourself using both models together. For example, you might use an SQL SELECT statement to create a small recordset of selected items from a large table, then use the Move navigational methods to step through the recordset and examine particular records one at a time.

Note The relational terms *row* and *column* are equivalent to the more familiar database terms *record* and *field*. Because SQL is a purely relational database language, the terms row and column are most frequently used to describe the operation of SQL commands. To avoid confusion, both sets of terms will be used interchangeably in this chapter. Just remember that a row is a record, and a column is a field.

The Components of SQL

The SQL language is composed of commands, clauses, operators, and aggregate functions. These elements are combined into statements used to create, update, and manipulate databases. The following sections briefly describe these elements of the SQL language, and the remainder of this chapter presents specific examples of their use.

Note The following sections present many of the most commonly used SQL commands and keywords, but not all of them. For a complete reference listing of SQL keywords, search for "SQL" in Books Online.

SQL Commands

Like the DAO navigational methods, SQL provides both data definition language (DDL) and data manipulation language (DML) commands. Although there are some areas of overlap, the DDL commands allow you to create and define new databases, fields, and indexes, while the DML commands let you build queries to sort, filter, and extract data from the database.

DDL

DDL statements in SQL are expressions built around the following commands.

Command	Description
CREATE	Used to create new tables, fields, and indexes.
DROP	Used to delete tables and indexes from the database.
ALTER	Used to modify tables by adding fields or changing field definitions.

DML

DML statements are expressions built around the following commands.

Command	Description
SELECT	Used to query the database for records that satisfy specific criteria.
INSERT	Used to load batches of data into the database in a single operation.
UPDATE	Used to change the values of particular records and fields.
DELETE	Used to remove records from a database table.

SQL Clauses

Clauses are modifying conditions used to define the data you want to select or manipulate. The following table lists the clauses you can use.

Clause	Description
FROM	Used to name the table from which records are to be selected.
WHERE	Used to specify the condition(s) the records must meet to be selected.
GROUP BY	Used to separate the selected records into specific groups.
HAVING	Used to state the condition to be satisfied by each group.
ORDER BY	Used to sort the selected records according to a specified order.

SQL Operators

There are two kinds of operators in SQL: logical operators and comparison operators.

Logical Operators

Logical operators are used to connect expressions, usually within a WHERE clause. For example:

```
SELECT * from Mytable WHERE condition1 AND condition2
```

Here the AND operator connects the expressions `condition1` and `condition2`, to specify that both conditions must be met to satisfy the selection criteria. The logical operators include:

- AND
- OR
- NOT

Comparison Operators

Comparison operators are used to compare the relative value of two expressions to determine what action should be taken. For example:

```
SELECT * from Publishers WHERE PubId = 5
```

Here the '=' operator specifies that only those records that have a PubID field with a value of 5 will be selected.

The comparison operators are listed in the following table.

Operator	Meaning/Usage
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
=	Equal to
<>	Not equal to
BETWEEN	Used to specify a range of values
LIKE	Used in pattern matching
IN	Used to specify records in a database

Aggregate Functions

Aggregate functions are used within a SELECT clause on groups of records to return a single value that applies to a group of records. For example, the AVG aggregate function can return the average of all the values in a particular field of a recordset. The following table lists the aggregate functions.

Aggregate function	Description
AVG	Used to obtain the average of the values in a particular field.
COUNT	Used to return the number of records in the selection.
SUM	Used to return the sum of all the values in a particular field.
MAX	Used to return the highest value in a specified field.
MIN	Used to return the smallest value in a specified field.

DDL Operations

The SQL data definition language (DDL) includes a number of commands you can use to create tables and indexes, and modify tables by adding or removing columns or indexes. These data definition statements can only be used with Jet databases; they are not supported for any of the external database formats.

Note To use the DDL commands, or any query that doesn't return rows of records, enclose the entire statement in double quotes and use it as the argument to the Execute method of a Database or QueryDef object, as in:

```
MyDB.Execute "CREATE TABLE Employees _
([First Name] TEXT, [Last Name] TEXT)"
```

To use any of the commands that return rows (such as SELECT), use the statement as the source argument to the OpenRecordset method, as in:

```
MyDB.OpenRecordset("SELECT * FROM Titles WHERE _
Au_ID = 5",dbOpenDynaset)
```

Creating a Table

To create new tables in a database, use the CREATE TABLE statement. The complete statement accepts arguments for the table name and, for each column (field), you add a set of arguments for the field name, data type, and, for text columns, size in characters.

The following example creates a table called "Employees," having two text columns of 25 characters each:

```
CREATE TABLE Employees ([First Name] TEXT (25), _
[Last Name] TEXT (25))
```

Adding and Removing Columns

You can add, modify, or remove columns in a table with the ALTER TABLE statement. For example, the following statement adds a 25-character Text column named "Notes" to the Employees table:

```
ALTER TABLE Employees ADD COLUMN Notes TEXT(25)
```

To remove a column, use the DROP keyword. This example removes the "Notes" column added in the previous example:

```
ALTER TABLE Employees DROP COLUMN Notes
```

To modify a column, you must first delete it, and then add a new column of the same name. The following example increases the size of the Notes column by dropping it from the table, and then adding a new, longer Notes field:

```
ALTER TABLE Employees DROP COLUMN Notes;
ALTER TABLE Employees ADD COLUMN Notes TEXT(30);
```

Note Using ALTER TABLE, you can add or delete only one column at a time.

Creating and Deleting Indexes

There are three different ways to create an index:

- At table creation, with the CREATE TABLE statement.
- With the CREATE INDEX statement.
- With the ALTER TABLE statement.

Although all three approaches can be used to create similar indexes, there are some differences. To add a foreign key and enforce referential integrity, you must use a CONSTRAINT clause in a CREATE TABLE or ALTER TABLE statement.

Sometimes it is preferable to create a table without an index first, and then design the index parameters after using the prototype table. In this situation, you would use CREATE TABLE to create the prototype table with no index, and then later on add the index with CREATE INDEX or ALTER TABLE.

Creating an Index with CREATE TABLE

When you create a table, you can create an index on individual columns, or across two or more columns, using the SQL CONSTRAINT clause (the CONSTRAINT keyword begins the definition of an index). The following example shows how to create a table with a 3-column index:

```
CREATE TABLE Employees ([First Name] TEXT (25), _
[Last Name] TEXT (25), [Date of Birth] DATETIME, _
CONSTRAINT EmployeesIndex UNIQUE _
([First Name], [Last Name], [Date of Birth]));
```

To index only one column, you put the CONSTRAINT clause in one of the column definitions. For example, to index the Date of Birth column only, you would use the following CREATE TABLE statement:

```
CREATE TABLE Employees ([First Name] TEXT (25), _
[Last Name] TEXT (25), [Date of Birth] DATETIME _
CONSTRAINT EmployeesIndex PRIMARY);
```

The difference between a multiple-column index and a single-column index is that, for a single-column index, the CONSTRAINT keyword that begins the index definition is not separated from the last column by a comma, but is instead immediately after the data type of the indexed column.

Creating an Index with CREATE INDEX

You can also use the CREATE INDEX statement to add an index. This example yields the same results as the previous example, except it uses CREATE INDEX instead of ALTER TABLE:

```
CREATE UNIQUE INDEX MyIndex ON Employees _
([Date of Birth])
```

In the optional WITH clause, you can enforce data validation rules: PRIMARY, meaning this is the primary index column; DISALLOW NULL, meaning this column may not be left blank; or IGNORE NULL, meaning that the record will not be indexed if this column is blank.

The following example adds a WITH clause to the previous example, so that no record may be added to the table with an empty social security number column:

```
CREATE UNIQUE INDEX MyIndex ON Employees (SSN) _
WITH DISALLOW NULL
```

Note Don't use the PRIMARY keyword when you create a new index on a table that already has a primary key; if you do, an error occurs.

You can use CREATE INDEX to create an index specification on an attached table that does not already have an index. To create such an index, you don't need permission or access to the remote server, and the remote database is unaware of and unaffected by the index. You use the same syntax for attached and native tables. This can be especially useful to create an index on a table that would ordinarily be read-only because it lacks an index.

Creating an Index with ALTER TABLE

You can also add an index to an existing table using the ALTER TABLE statement, using the ADD CONSTRAINT syntax. For example, this statement adds an index to the "SSN" column of a table:

```
ALTER TABLE Employees ADD CONSTRAINT MyIndex _
PRIMARY (SSN)
```

You can also add a multiple-column index to a table using the ALTER TABLE statement, like this:

```
ALTER TABLE Employees ADD CONSTRAINT NameIndex _
UNIQUE ([Last Name], [First Name], SSN)
```

The CONSTRAINT Clause and Referential Integrity

A constraint is an index. You use the CONSTRAINT clause to create or delete indexes with the CREATE TABLE and ALTER TABLE statements, as shown in the preceding section.

The CONSTRAINT clause also allows you to define primary and foreign keys, to define relations and enforce referential integrity.

For More Information For information about relations and referential integrity, see Chapter 2, "Creating and Modifying Databases."

There are two types of CONSTRAINT clauses: one for creating an index on a single field and one for creating an index on more than one field.

The syntax for the single-field index is:

CONSTRAINT *name* {**PRIMARY KEY** | **UNIQUE** | **REFERENCES** foreigntable [(*foreignfield1*, *foreignfield2*)]}

The syntax for a multiple-field index is:

CONSTRAINT *name* {**PRIMARY KEY** (*primary1*[, *primary2* [, ...]]) |**UNIQUE** (*unique1*[, *unique2* [, ...]]) |**FOREIGN KEY** (*ref1*[, *ref2* [, ...]]) **REFERENCES** *foreigntable* [(*foreignfield1* [, *foreignfield2* [, ...]])]}

The following arguments apply to both versions.

Argument	Description
name	The name of the index to be created.
primary1, primary2	The name of the field or fields to be designated the primary key.
unique1, unique2	The name of the field or fields to be designated as a unique key.
ref1, ref2	The name of a foreign key field or fields that refer to fields in another table.
foreigntable	The name of the foreign table containing the field or fields specified by foreignfield.
foreignfield1, foreignfield2	The name of the field or fields in foreigntable specified by ref1, ref2.

Using CONSTRAINT, you can designate a field as one of the following types of indexes:

- UNIQUE—designates a field as a unique key. This means that no two records in the table can have the same value in this field. You can constrain any field or list of fields as unique. If a multiple-field index is designated as a unique key, the combined values of all fields in the index must be unique, even if two or more records have the same value in just one of the fields.
- PRIMARY KEY—designates one field or set of fields in a table as a primary key. All values in the primary key must be unique, and there can be only one primary key for a table. If you set a PRIMARY KEY constraint on a table that already has a primary key, an error occurs.
- FOREIGN KEY—designates a field as a foreign key. If the foreign table's primary key consists of more than one field, you must use a multiple-field index definition, listing all of the referencing fields, the name of the foreign table, and the names of the referenced fields in the foreign table in the same order that the referencing fields are listed. If the referenced field or fields are the foreign table's primary key, you don't have to specify the referenced fields by default—the Jet engine behaves as if the foreign table's primary key is the referenced fields.

For example, to add the PubID index for the Titles table of the Biblio.mdb sample database, you could use the following statement:

```
ALTER TABLE Titles ADD CONSTRAINT MyIndex _
FOREIGN KEY (PubID) REFERENCES Publishers (PubID)
```

Notice that by using the FOREIGN KEY keywords, you are creating a relation between the PubID field of the Titles table (foreign key) and the PubID field of the Publishers table (primary key). This relation will be enforced by the Jet engine, just as if you had used the CreateRelation method described in Chapter 2, "Creating and Modifying Databases."

DML Operations

The SQL data manipulation language (DML) statements are used to retrieve records in tables, update records, and add or delete records in tables. A number of different statements are supported for these tasks, but most of them fall within the general structure of the SELECT query.

The Select Query

Use the SELECT statement to retrieve records from the database as a set of records, storing them in a new Recordset object. Your application can then manipulate this recordset—displaying, adding, changing, and/or deleting records as needed. Your application can also display and produce reports from the data.

SELECT is usually the first word in an SQL statement. Most SQL statements are either SELECT or SELECT...INTO statements. You can use a SELECT statement in the SQL property of a QueryDef object, in the RecordSource property of a data control, or as an argument for the OpenRecordset method. SELECT statements don't change data in the database; they only retrieve data.

The overall form of the SELECT query is as follows:

SELECT *fieldlist*
FROM *tablenames* **IN** *databasename*
WHERE *searchconditions*
GROUP BY *fieldlist*
HAVING *group criteria*
ORDER BY *fieldlist*
WITH OWNERACCESS OPTION

Each of these statements and clauses will be discussed in the following sections.

The Basic Query

The simplest possible SELECT query is

SELECT * FROM *tablename*

For example, the following SELECT query will return all columns of all records in the Employees table:

```
SELECT * FROM Employees
```

The asterisk indicates that all columns of the desired table(s) are to be retrieved. You could specify only certain columns. When displayed, data for each column will appear in the order in which they are listed, so you can re-order columns for readability:

```
SELECT [First Name], [Last Name] FROM Employees
```

Specifying the Data Source

A SELECT statement will always have a FROM clause, indicating the table or tables from which records will be drawn.

If a field name is included in more than one table in the FROM clause, precede it with the table name and the . (dot) operator. In the following example, the Department field is in both the Employees table and the Supervisors table. The SQL statement selects Department from the Employees table and SupvName from the Supervisors table:

```
SELECT Employees.Department, SupvName _
FROM Supervisors, Employees _
WHERE Employees.Department = Supervisors.Department;
```

When the FROM clause lists more than one table, the order in which they appear is not important.

Specifying a Table in an External Database

Sometimes, you will need to reference a table in an external database to which the Microsoft Jet database engine can connect, such as a dBASE or Paradox database or an external Jet database. You can do this with the optional IN clause. The IN clause usually appears following a table name in a FROM clause, but can also be used in a SELECT INTO or INSERT INTO clause, where the target table is in an external database.

Note You can use IN to connect to only one external database at a time.

In some cases, the path argument refers to the directory containing the database files. For example, when working with dBASE, FoxPro, or Paradox database tables, the *path* argument specifies the directory containing .DBF or .DB files. The table filename is derived from the *destination* or *tableexpression* arguments.

To specify a non-Jet database, append a semicolon (;) to the name, and enclose it in single (') or double (") quotation marks. For example:

```
'dBASE IV;'
```

You can also use the DATABASE keyword to specify the external database. For example, both of the following lines specify the same table:

```
SELECT * FROM Table IN "" [dBASE IV; _
DATABASE=C:\DBASE\DATA\SALES;];
SELECT * FROM Table IN "C:\DBASE\DATA\SALES" _
"dBASE IV;"
```

Note For improved performance and ease of use, it is almost always better to use an attached table instead of an IN clause.

For More Information For more information about attached tables, see Chapter 3, "Working with Records and Fields," and Chapter 7, "Accessing External Data."

Aliasing the Column Names

When a Recordset object is created from a SELECT statement, the table's column names become the Field object names in the Recordset. If you want different column names, use the AS clause. The following example uses the title "DOB" to name the returned Birth Date column from the Employees table:

```
SELECT [Birth Date] AS DOB FROM Employees;
```

Whenever you use queries that return ambiguous or duplicate Field object names, you must use the AS clause to provide an alternate name for the Field. The following example uses the title "Head Count" to name the returned Field in the resulting Recordset:

```
SELECT COUNT(Employee ID) AS [Head Count] _
FROM Employees;
```

Using Visual Basic Variables in an SQL Statement

In a Visual Basic program, you can create a SELECT statement in your application by concatenating local variables into the statement as needed to select, sort, or filter the data as dictated by your application. For example, if you have a TextBox control (TitleWanted) containing the name of a Title and you want to fetch all books from the Titles table with this title, you can create an SQL statement that includes the current value of the TextBox. Note that the SQL query encloses the TitleWanted value in single quotation marks ('):

```
Set Rst = Db.OpenRecordset("SELECT * FROM Titles " _
& " WHERE Title = '" & TitleWanted.Text & "'" )
```

Filtering and Sorting the Query Results

SQL provides a number of optional keyword predicates and clauses that help you further refine a query and sort the result set. The most useful of these are discussed in the following sections.

The DISTINCT Predicate

To omit records that contain duplicate data in the selected columns, use the DISTINCT keyword. To be included in the results of the query, the values for each column or combination of columns listed in the SELECT statement must be unique. For example, several employees listed in an Employees table may have the same last name. If two records contain Smith in the Last Name field, the following SQL statement returns only one of the records:

```
SELECT DISTINCT _
   [Last Name] _
   FROM Employees;
```

If you omit DISTINCT, this query returns both Smith records.

The output Recordset of a DISTINCT query isn't updatable and doesn't reflect subsequent changes made by other users.

The TOP Predicate

To return only a certain number of records that fall at the top or the bottom of a range specified by an ORDER BY clause, use the TOP predicate. Suppose you want the names of the top 25 students from the class of 1994:

```
SELECT TOP 25 [First Name], [Last Name] FROM Students _
WHERE [Graduation Year] = 1994 _
ORDER BY [Grade Point Average] DESC;
```

If you don't include the ORDER BY clause, the query will return an arbitrary set of 25 records from the Students table that satisfy the WHERE clause.

The TOP predicate doesn't choose between equal values. In the preceding example, if the twenty-fifth and twenty-sixth highest ranked students have the same grade point average, the query will return 26 records.

You can also use the PERCENT keyword to return a certain percentage of records that fall at the top or the bottom of a range specified by an ORDER BY clause. Suppose that, instead of the top 25 students, you want the top 10 percent of the class:

```
SELECT TOP 10 PERCENT [First Name], [Last Name] _
FROM Students _
WHERE [Graduation Year] = 1994 _
ORDER BY [Grade Point Average] DESC;
```

The WHERE Clause

The WHERE clause specifies which records from the tables listed in the FROM clause are included in the results of the SELECT statement.

The Microsoft Jet database engine selects the records that meet the conditions listed in the WHERE clause. If you don't specify a WHERE clause, your query returns all rows from the table. If you specify more than one table in your query and you haven't included a WHERE clause or a JOIN clause, your query generates a Cartesian product of the tables.

Note Although a WHERE clause can accomplish similar tasks, you must use a JOIN clause to perform SQL join operations on multiple tables if you want the resulting recordset to be updatable.

WHERE is optional, but when included, follows FROM. For example, you can select all employees in the sales department

```
WHERE Dept = 'Sales'
```

or all customers between the ages of 18 and 30

```
WHERE Age Between 18 AND 30
```

WHERE is similar to HAVING. WHERE determines which records are selected. Similarly, once records are grouped with GROUP BY, HAVING determines which records are displayed.

Use the WHERE clause to eliminate records you don't want grouped by a GROUP BY clause.

Use various expressions to determine which records the SQL statement returns. For example, the following SQL statement selects all employees whose salaries are more than $21,000:

```
SELECT [Last Name], Salary _
FROM Employees _
WHERE Salary > 21000;
```

A WHERE clause can contain up to 40 expressions linked by logical operators, such as AND and OR.

When you enter a field name that contains a space or punctuation, surround the name with brackets ([ ]):

```
SELECT [Product ID], [Units In Stock] P
FROM Products _
WHERE [Units In Stock] <= [Reorder Level];
```

When you specify the *criteria* argument, date literals must be in U.S. format, even if you're not using the U.S. version of the Jet database engine. For example, May 10, 1994, is written 10/5/94 in the United Kingdom and 5/10/94 in the United States. Be sure to enclose your date literals with the number sign (#), as shown in the following examples.

To find records dated May 10, 1994 in a United Kingdom database, you must use the following SQL statement:

```
SELECT * FROM Orders _
WHERE [Shipped Date] = #5/10/94#;
```

You can also use the DateValue function, which is aware of the International settings established by Microsoft Windows. For example, for code in the United States:

```
SELECT * FROM Orders _
WHERE [Shipped Date] = DateValue('5/10/94');
```

For code in the United Kingdom:

```
SELECT * FROM Orders _
WHERE [Shipped Date] = DateValue('10/5/94');
```

For More Information For more information about international issues, see Chapter 16, "International Issues," in the *Programmer's Guide.*

GROUP BY Clause

GROUP BY is an optional clause that combines records with identical values in the specified field list into a single record. A summary value is created for each record if you include an SQL aggregate function, such as Sum or Count, in the SELECT statement.

Summary values are omitted if there is no SQL aggregate function in the SELECT statement.

Null values in GROUP BY fields are grouped and aren't omitted. However, Null values aren't evaluated in any SQL aggregate function.

Use the WHERE clause to exclude rows you don't want grouped, and use the HAVING clause to filter records after they've been grouped.

Unless it contains Memo or Automation object data, a field in the GROUP BY field list can refer to any field listed in the FROM clause, even if the field isn't included in the SELECT statement, provided the SELECT statement includes at least one SQL aggregate function. The Jet database engine can't group on Memo or Automation object fields.

All fields in the SELECT field list must either be included in the GROUP BY clause or be included as arguments to an SQL aggregate function.

When you enter a field name that contains a space or punctuation, surround the name with brackets ([]):

```
SELECT [Product Name], Sum([Units in Stock]) _
FROM Products _
GROUP BY [Product Name];
```

HAVING Clause

Specifies which grouped records are displayed in a SELECT statement with a GROUP BY clause. Once GROUP BY combines records, HAVING displays any records grouped by the GROUP BY clause that satisfy the conditions of the HAVING clause.

HAVING is similar to WHERE, which determines which records are selected. Once records are grouped with GROUP BY, HAVING determines which records are displayed.

HAVING is optional. A HAVING clause can contain up to 40 expressions linked by logical operators, such as AND and OR.

ORDER BY Clause

The ORDER BY clause determines the sort order of records retrieved in the query. In the ORDER BY clause, you specify the column or columns to be used as the ordering key, and then specify whether records are to appear in ascending or descending order of magnitude. The following example returns all the records in the Employees table, listed by last name in alphabetical order:

```
SELECT * FROM Employees _
ORDER BY [Last Name] ASC;
```

In this example, the ASC (for "ascending order") is optional—the default sort order is ascending (A–Z, 0–9). However, you may want to include the ASC keyword at the end of each field you want to sort in ascending order, to clearly document the ORDER BY clause.

Both of the following examples sort employee names in last name order:

```
SELECT [Last Name], [First Name] FROM Employees _
ORDER BY [Last Name];
```

```
SELECT [Last Name], [First Name] FROM Employees _
ORDER BY [Last Name] ASC;
```

To sort in descending order (Z–A, 9–0), add the DESC keyword to the end of each field you want to sort in descending order.

You can also use the ordinal number of the ordering column as it appears in the SELECT list, instead of retyping its name:

```
SELECT [First Name], [Last Name] FROM Employees _
ORDER BY 2 ASC;
```

You can include additional fields in the ORDER BY clause. Records are sorted first by the first field listed after ORDER BY. Records that have equal values in that field are then sorted by the value in the second field listed, and so on. The following example selects salaries and sorts them in descending order; all employees having the same salary are listed alphabetically in ascending order:

```
SELECT [Last Name], Salary FROM Employees _
ORDER BY Salary DESC, [Last Name];
```

ORDER BY is usually the last item in an SQL statement. It is optional (unless you use the TOP or TOP n PERCENT predicate in the SELECT clause). If you don't include ORDER BY, the data is displayed unsorted.

WITH OWNERACCESS OPTION Statement

In a multiuser environment with a secure workgroup, use WITH OWNERACCESS OPTION at the end of a query to give the user who runs the query permission to view the data in the query even if the user is otherwise restricted from viewing the query's underlying tables.

The following example enables the user to return salary information, even if the user doesn't otherwise have permission to view the Payroll table, provided the query is created by someone who has that permission:

```
SELECT [LastName], [FirstName], Salary FROM Employees _
ORDER BY [LastName] _
WITH OWNERACCESS OPTION;
```

If a user is otherwise prevented from creating or adding to a table, you can use WITH OWNERACCESS OPTION to enable the user to run a make-table or append query.

This option requires you to have access to the System.mda file associated with the database. It's really useful only in secured multiuser implementations.

Using a Make-Table Query

A variation on the SELECT statement lets you create a new table, instead of a Recordset object, with the returned records. To do this, you add the INTO clause. The following example creates a new table called New Employees by querying the Employees table:

```
SELECT * INTO [New Employees] FROM Employees
```

You can use make-table queries to archive records, make backup copies of your tables, or make copies to export to another database or to use as a basis for reports that display data for a particular time period. For example, you could produce a Monthly Sales by Region report by running the same make-table query each month.

You may want to define a primary key for the new table. When you create the table, the fields in the new table inherit the data type and field size of each field in the query's underlying tables, but no other field or table properties are transferred.

Using a Delete Query

Creates a delete query that removes records from one or more of the tables listed in the FROM clause that satisfy the WHERE clause, as shown in the following syntax:

DELETE [*table*.*]
  FROM *tableexpression*
  WHERE *criteria*

DELETE is especially useful when you want to delete many records.

In a multiple-table DELETE statement, you must include the *table* argument. If you specify more than one table from which to delete records, none of the tables may contain the primary key of a one-to-many relationship.

If you want to delete all of the records in a table, deleting the table itself may be more efficient than running a delete query. You can use the Execute method with a DROP TABLE statement to drop a table from the database. If you delete the table, however, the structure is lost. In contrast, when you use DELETE, only the data is deleted; the table structure and all of the table properties, such as field attributes and indexes, remain intact.

You can use DELETE to remove records from a single table or from the many side of a one-to-many relationship. Cascade delete operations in a query only delete from the many side of a relationship. For example, in the relationship between the Customers and Orders tables, the Orders table is the many side, so cascade delete operations would only affect the Orders table.

A delete query deletes entire records, not just data in specific fields. If you want to delete values in a specific field, create an update query that changes the values to Null.

Once you remove records using a delete query, you can't undo the operation. If you want to know which records were deleted, first examine the results of a select query that uses the same criteria, and then run the delete query.

You should maintain backup copies of your data at all times. If you accidentally delete the wrong records, you can retrieve them from your backup copies.

Using an Append Query

You can use the INSERT INTO statement to add records to a table or create an append query.

You can use the following syntax to perform a multiple-record append query:

INSERT INTO *target* [**IN** *externaldatabase*]
  SELECT [*source.*]*field1*[, *field2*[, ...]
  FROM *tableexpression*

Conversely, use this syntax to perform a single-record append query:

INSERT INTO *target* [(*field1*[, *field2*[, ...]])]
  VALUES (*value1*[, *value2*[, ...])

You can use the INSERT INTO statement to add a single record to a table using the single-record append query syntax. In this case, your code specifies the name and value for each field of the record. You must specify each of the fields of the record to which a value will be assigned as well as a value for that field. When you don't specify each field, the default value or Null is inserted for missing columns. Records are added to the end of the table.

You can also use INSERT INTO to append a set of records from another table or query by using the SELECT...FROM clause as shown in the multiple-record append query syntax. In this case, the SELECT clause specifies the fields to append to the *target* table specified.

The source or target table may specify a table or a query. If a query is specified, the Microsoft Jet database engine appends a set of records to any and all tables.

INSERT INTO is often used to append a new table of customer records to your active Customers table.

INSERT INTO is optional, but when included, precedes the SELECT statement.

If your destination table contains a primary key, make sure you append unique, non-Null values to the primary key field or fields; if you don't, the Jet engine won't append the records.

If you append records to a table with a Counter field, don't include the Counter field in your query if you want the Jet engine to renumber the appended records. Include the Counter field in the query if you want to retain the original values from the field. If there are duplicate values, however, the Jet engine won't append the records.

Use the IN clause to append records to a table in another database.

To create a new table, use the SELECT...INTO statement instead to create a make-table query.

To find out which records will be appended before you run the append query, first execute and view the results of a select query that uses the same selection criteria.

An append query copies records from one or more tables to another. The tables that contain the records you append aren't affected by the append query.

Instead of appending existing records from another table, you can specify the value for each field in a single new record using the VALUES clause. If you omit the field list, the VALUES clause must include a value for every field in the table; otherwise, the INSERT will fail. Use an additional INSERT INTO statement with a VALUES clause for each additional record you want to create.

Using an Update Query

UPDATE creates an update query that changes values in fields in a specified table based on specified criteria.

UPDATE *table*
SET *newvalue*
WHERE *criteria*;

UPDATE is especially useful when you want to change many records or when the records that you want to change are in multiple tables. Generally, you use the UPDATE statement with the Execute method.

You can change several fields at the same time. The following example increases the Order Amount values by 10 percent and the Freight values by 3 percent for shippers in the United Kingdom:

```
UPDATE Orders _
SET [Order Amount] = [Order Amount] * 1.1, _
Freight = Freight * 1.03 _
WHERE [Ship Country] = 'UK';
```

UPDATE doesn't generate a result set. If you want to know which records will be changed, first examine the results of a select query that uses the same criteria, and then run the update query.

Complex Queries

Complex queries combine one or more SELECT statements, or multiple FROM clauses within a SELECT statement, to perform queries that cannot be accomplished by a single query against a single set of tables.

Subqueries

A subquery is a SELECT statement nested inside a SELECT, SELECT...INTO, INSERT...INTO, DELETE, or UPDATE statement, or inside another subquery.

You can use three forms of syntax to create a subquery:

comparison [**ANY** | **ALL** | **SOME**] (*sqlstatement*)
expression [**NOT**] **IN** (*sqlstatement*)
[**NOT**] **EXISTS** (*sqlstatement*)

You can use a subquery instead of an expression in the field list of a SELECT statement or in a WHERE or HAVING clause. In a subquery, you use a SELECT statement to provide a set of one or more specific values to evaluate in the WHERE or HAVING clause expression.

Use the ANY or SOME predicate, which are synonymous, to retrieve records in the main query that satisfy the comparison with any records retrieved in the subquery.

The following example returns all products whose unit price is greater than that of any product sold at a discount of 25 percent or more:

```
SELECT * FROM Products _
WHERE [Unit Price] > ANY _
(SELECT [Unit Price] FROM [Order Details] _
WHERE [Discount] >= 25);
```

Use the ALL predicate to retrieve only those records in the main query that satisfy the comparison with all records retrieved in the subquery. If you changed ANY to ALL in the preceding example, the query would return only those products whose unit price is greater than that of all products sold at a discount of 25 percent or more. This is much more restrictive.

Use the IN predicate to retrieve only those records in the main query for which some record in the subquery contains an equal value. The following example returns all products sold at a discount of 25 percent or more:

```
SELECT * FROM Products _
WHERE [Product ID] IN _
(SELECT [Product ID] FROM [Order Details] _
WHERE [Discount] >= 25);
```

Conversely, you can use NOT IN to retrieve only those records in the main query for which no record in the subquery contains an equal value.

Use the EXISTS predicate (with the optional NOT keyword) in true/false comparisons to determine whether the subquery returns any records.

You can also use table name aliases in a subquery to refer to tables listed in a FROM clause outside the subquery. In the following example, the Employees table is given the alias "T1":

```
SELECT [Last Name], [First Name], Title, Salary _
FROM Employees AS T1 _
WHERE Salary >= _
(SELECT Avg(Salary) _
FROM Employees _
WHERE T1.Title = Employees.Title) Order by Title;
```

Crosstab Queries

Crosstab queries let you select values from specified fields or expressions as column headings, so you can view your data in a more compact format than with a normal SELECT query. You use the TRANSFORM statement to create crosstab queries:

TRANSFORM *aggfunction*
selectstatement
PIVOT *pivotfield* [**IN** (*value1*[, *value2*[, ...]])]

The **TRANSFORM** statement uses the following arguments.

Argument	Description
aggfunction	An SQL aggregate function that operates on the selected data.
selectstatement	A SELECT statement.
pivotfield	The field or expression you want to use to create column headings in the query's result set.
value1, value2	Fixed values used to create column headings. When you summarize data using a crosstab query, you select values from specified fields or expressions as column headings so you can view data in a more compact format than with a select query.

TRANSFORM precedes a SELECT statement that specifies the fields used as row headings and a GROUP BY clause that specifies row grouping. Optionally, you can include other clauses, such as WHERE, that specify additional selection or sorting criteria.

The values returned in *pivotfield* are used as column headings in the query's result set. For example, pivoting the sales figures on the month of the sale in a crosstab query would create 12 columns. You can restrict *pivotfield* to create headings from fixed values (*value1*, *value2*) listed in the *optional* IN clause. You can also include fixed values for which no data exists to create additional columns.

The following example creates a crosstab query that shows product sales by month for a user-specified year. The months are returned from left to right (pivoted) as columns, and the product names are returned from top to bottom as rows.

```
PARAMETERS [Sales for which year?] LONG; TRANSFORM _
Sum([Order Details].Quantity * ([Order Details]. _
[Unit Price] - ([Order Details].Discount / 100) _
* [Order Details].[Unit Price])) AS Sales SELECT _
[Product Name] FROM Orders INNER _
JOIN(Products INNER JOIN [Order Details] ON _
Products.[Product ID] = [Order Details]._
[Product ID]) ON Orders.[Order ID] = _
[Order _ Details].[Order ID] WHERE DatePart_
("yyyy", [Order Date]) = [Sales for which year?] _
GROUP BY [Product Name] ORDER BY [Product Name] _
PIVOT DatePart("m", [Order Date])
```

Joins

One of the most powerful characteristics of relational databases is the ability to join two (or more) tables together to create a new table (or recordset) containing information from both tables.

Tables are joined according to relations between them, most commonly between the primary key of one table and the corresponding foreign key of another table. Depending on how the tables are joined, you can create the following kind of joins.

Join	Description
INNER JOIN	Records from both tables are included in the join only when a specified field in the first table matches the specified field from the second table.
LEFT OUTER JOIN	All the records from the first table are included in the join, together with records from the second table where the specified fields match.
RIGHT OUTER JOIN	All the records from the second table are included in the join, together with records from the first table where the specified fields match.

Inner Joins

To create a query that includes only records in which the data in the joined fields is the same, use an INNER JOIN operation.

INNER JOIN combines records from two tables whenever there are matching values in a common field. Use the following syntax:

FROM *table1* **INNER JOIN** *table2* **ON** *table1.field1* = table2.*field2*

You can use an INNER JOIN operation in any FROM clause. It creates an *equi-join*, which is also known as an inner join. Equi-joins are the most common type of join. They combine records from two tables whenever there are matching values in a field common to both tables.

You can use INNER JOIN with the Departments and Employees tables to select all the employees in each department. In contrast, to select all departments (even if some have no employees assigned to them) or all employees (even if some aren't assigned to a department), you can use a LEFT JOIN or RIGHT JOIN operation to create an outer join.

You can join any two numeric fields, even if they are of different data types. For example, you can join a Number field, for which the Size property of its Field object is set to Integer, and a Counter field.

The following example shows how you could join the Categories and Products tables on the Category ID field:

```
SELECT [Category Name], [Product Name] _
FROM Categories INNER JOIN Products _
ON Categories.[Category ID] = Products.[Category ID];
```

In the preceding example, Category ID is the joined field, but it isn't included in the query output because it isn't included in the SELECT statement. To include the joined field, include the field name in the SELECT statement — in this case, `Categories.[Category ID]`.

The following example creates two equi-joins: one between the Order Details and Orders tables and another between the Orders and Employees tables. This is necessary because the Employees table doesn't contain sales data, and the Order Details table doesn't contain employee data. The query produces a list of employees and their total sales.

```
Dim MyQRY As QueryDef
MyQRY.SQL = "SELECT DISTINCTROW Sum([Unit Price] _
[Quantity]) AS [Sales], [First Name] & " " & _
[Last Name] AS Name FROM Employees _
INNER JOIN (Orders INNER JOIN [Order Details] _
ON Orders.[Order ID] = [Order Details].[Order ID]) _
ON Employees.[Employee ID] = Orders.[Employee ID] _
GROUP BY [First Name] & " " & [Last Name];"
```

This statement must be used as the SQL property of a QueryDef object or in an SQL expression with the OpenRecordset method.

Omitting Duplicate Rows

The previous example uses the DISTINCTROW predicate in the SELECT clause to omit data based on entire duplicate records, not just duplicate fields. For example, you could create a query that joins the Customers and Orders tables on the Customer ID field. The Customers table contains no duplicate Customer ID fields, but the Orders table does because each customer can have many orders. The following SQL statement shows how you can use DISTINCTROW to produce a list of companies that have at least one order but without any details about those orders:

```
SELECT DISTINCTROW [Company Name] _
FROM Customers INNER JOIN Orders _
ON Customers.[Customer ID] = Orders.[Customer ID] _
ORDER BY [Company Name];
```

If you omit DISTINCTROW, this query produces multiple rows for each company that has more than one order.

DISTINCTROW has an effect only when you select fields from some, but not all, of the tables used in the query. DISTINCTROW is ignored if your query includes only one table.

Left and Right (Outer) Joins

Outer joins combine source-table records when used in any FROM clause, with the following syntax:

FROM *table1* [**LEFT** | **RIGHT**] **JOIN** *table2*
  ON *table1.field1* = *table2.field2*

Use a LEFT JOIN operation to create a left outer join. Left outer joins include all of the records from the first (left-hand) of two tables, even if there are no matching values for records in the second (right-hand) table.

Use a RIGHT JOIN operation to create a right outer join. Right outer joins include all of the records from the second (right-hand) of two tables, even if there are no matching values for records in the first (left-hand) table.

For example, you could use LEFT JOIN with the Departments (left) and Employees (right) tables to select all departments, including those that have no employees assigned to them. To select all employees, including those who aren't assigned to a department, you would use RIGHT JOIN.

The following example shows how you could join the Categories and Products tables on the Category ID field. The query produces a list of all categories, including those that contain no products:

```
SELECT [Category Name], _
[Product Name] _
FROM Categories LEFT JOIN Products _
ON Categories.[Category ID] = Products.[Category ID];
```

In the following example, Category ID is the joined field, but it isn't included in the query results because it isn't included in the SELECT statement. To include the joined field, enter the field name in the SELECT statement—in this case, `Categories.[Category ID]`.

Nested Joins

You can also nest JOIN statements using the following syntax:

SELECT *fields*
  FROM *table1* **INNER JOIN**
  (*table2* **INNER JOIN** [( ]*table3*
  [**INNER JOIN** [( ]*tablex* [**INNER JOIN** ...)]
  ON *table3.field3* = *tablex.fieldx*)]
  ON *table2.field2* = *table3.field3*)
  ON *table1.field1* = *table2.field2*;

Union Queries

You use the UNION operation to create a union query, combining the results of two or more independent queries or tables.

query1 **UNION [ALL]** *query2* **[UNION [ALL]** *queryn* [...]]
The *query* placeholder represents a string expression identifying the field that contains the numeric data you want to average or an expression that performs a calculation using the data in that field. Operands in an expression can include the name of a table field, a constant, or a function (which can be either intrinsic or user-defined but not one of the other SQL aggregate or domain aggregate functions).

You can merge the results of a query and an SQL statement in a single UNION operation. The following example merges the results of an existing query named New Accounts and a SELECT statement:

```
TABLE [New Accounts] UNION ALL _
SELECT * _
FROM Customers _
WHERE [Order Amount] > 1000;
```

By default, no duplicate records are returned when you use a UNION operation; however, you can include the ALL predicate to ensure that all records are returned. This also makes the query run faster.

All queries in a UNION operation must request the same number of fields; however, the fields don't have to be of the same size or data type.

You can use a GROUP BY and/or HAVING clause in each *query* argument to group the returned data. You can use an ORDER BY clause at the end of the last *query* argument to display the returned data in a specified order.

Optimizing Queries

The subject of database optimization is very broad and inclusive. Many factors, including hardware and software configuration, Windows setup, and disk cache size, have nothing to do with queries *but* they can affect database performance.

Another important factor is using attached tables rather than direct connections to remote databases. This is discussed thoroughly in Chapter 7, "Accessing External Data." Yet another consideration is choosing the appropriate type of Recordset object for your particular use. Table-type, dynaset-type, and snapshot-type recordsets all have particular performance considerations, and these are discussed in Chapter 3, "Working with Records and Fields."

This section will briefly present a number of optimization tips that relate specifically to query construction, and to taking advantage of Jet's Rushmore™ query optimization in building SQL queries.

General Optimization Strategies

- Use field sorting sparingly, especially with nonindexed fields.
- Make sure tables you join from different sources are indexed or are joined on the primary key.
- If your data doesn't change often, use make-table queries to create tables from your query dynasets. Use the resulting tables rather than queries as the basis for your forms, reports, or other queries.

- If you are creating a crosstab query, use fixed column headings whenever possible.
- Use the BETWEEN...AND, the IN, and the equal (=) operators on indexed columns.
- When creating a query, don't add fields that you don't need.

Optimizing Queries with Rushmore Technology

Rushmore is a data-access technology used in the Jet database engine that permits sets of records to be queried very efficiently. With Rushmore, when you use certain types of expressions in query criteria, your query will run much faster.

Simple Optimizable Expressions

The Jet database engine can optimize simple expressions in a WHERE clause in an SQL SELECT statement. A simple optimizable expression can form an entire expression or can appear as part of an expression.

A simple optimizable expression takes one of the following forms:

indexedfield comparisonoperator expression
–or–
expression comparisonoperator indexedfield

In a simple optimizable expression:

- *Indexedfield* must be a field on which an index is constructed.
- *Comparisonoperator* must be one of the following: <, >, =, <=, >=, <>, BETWEEN, LIKE, or IN.
- *Expression* may be any valid expression, including constants, functions, and fields from other tables.

Note For best results, the comparison value in an expression using the LIKE operator must begin with a character, not a wildcard. For example, you can optimize `LIKE "m*"` but not `LIKE "*m*"`.

Complex Optimizable Expressions

Microsoft Jet uses Rushmore to optimize complex expressions made by combining simple optimizable expressions with the AND or OR operators. A complex optimizable expression takes one of the following forms:

simpleexpression **AND** *simpleexpression*
–or–
simpleexpression **OR** *simpleexpression*

Keep the following in mind when using Rushmore optimizable expressions:

- The COUNT(*) function is highly optimized for queries that use Rushmore.
- If the index is descending and the comparison operator is other than equal (=), the query can't be optimized.
- Rushmore queries will work with Microsoft Access tables, as well as Microsoft FoxPro and dBASE tables (.DBF files). You can't use Rushmore with ODBC data sources, because DAO sends these queries to the ODBC data source instead of processing them locally.
- You can optimize multiple-field indexes if you query the indexed fields in the order they appear in the Indexes.Fields collection, beginning with the first indexed field and continuing with adjacent fields (up to and including 10 fields). For example, if you have a multiple-field index on Last Name, First Name, you can optimize a query on Last Name or on Last Name and First Name, but you can't optimize a query on First Name.

Differences Between Jet and ANSI SQL

Microsoft Jet database engine SQL is generally ANSI-89 Level 1 compliant. However, certain ANSI SQL features aren't implemented in Jet SQL. Conversely, Jet SQL includes keywords and features not supported in ANSI SQL.

Major Differences

The following is a list of the major differences between Jet SQL and ANSI SQL:

- Jet SQL and ANSI SQL each have different keywords and data types.
- Different rules apply to the BETWEEN...AND construct, which has the following syntax:

  expression1 **[NOT] BETWEEN** *expression2* **AND** *expression3*

  In Jet SQL, *expression2* can be greater than *expression3*; in ANSI SQL, *expression2* must be equal to or less than *expression3*.
- Different wildcard characters are used with the LIKE predicate.

Matching character	Jet SQL	ANSI SQL
Any single character	?	_ (underscore)
Zero or more characters	*	%

- Jet SQL is generally less restrictive. For example, it permits grouping and ordering on expressions.

Enhanced Features of Jet SQL

Jet SQL provides the following enhanced features:

- The TRANSFORM statement, which provides support for crosstab queries.
- Additional aggregate functions, such as StDev and VarP.
- The PARAMETERS declaration for defining parameter queries.

CHAPTER 6

Database Replication

If you design your applications for multiple users, database replication can improve the way your users share data. Using database replication, you can reproduce a database so that two or more users can work on their own replica of the database at the same time. Although you can place replicas on different computers or in different offices, they remain synchronized.

This chapter discusses how to create and use replicas of your Microsoft Jet application database.

What Is Database Replication?

Database replication is the technology for copying a database so that two or more copies remain synchronized. Each copy of the database is called a *replica*, and each replica contains a common set of tables, queries, forms, reports, macros, and modules; each replica can also contain *local objects* that exist only on a user's local computer.

Each replica is part of a *replica set*, which contains the replicas and the Design Master for a database. The *Design Master* is the only replica where you can make design changes to the database. Replicas that belong to the same replica set can exchange updates of data or replicated objects. This exchange is called *synchronization*.

Why Use Database Replication?

Imagine that a client has asked you to develop a contact management application for the company's field sales staff to use to monitor sales and orders. Each sales representative has a laptop computer that can be connected to the company's network.

Traditionally, building this application meant separating the tables from the other objects in the database so that the data can reside on a network server while the queries, forms, reports, macros, and modules reside on a user's computer. When sales representatives want to retrieve or update information in the database, they log on to the network, open a ContactForm database on the computer, and open the ContactData database from the server.

With database replication, you can create a single database that contains both the data and objects, and then make replicas of the database for each sales representative.

Or you can use linked tables and replicate the data database and application database separately. You can make replicas on each user's computer and synchronize each replica with the replica on a network server. Sales representatives update the replicas on their computers during the course of a work session, and then you synchronize their replicas with the replica on the server, as needed. You can also create a set of local objects, such as tables, forms, or reports, that a user or set of users access at only one replica location.

You should use database replication if you need to:

- Share data among offices. You can use database replication to create copies of a corporate database to send to each satellite office. Each location enters data into its replica, and all remote replicas are synchronized with the replica at corporate headquarters. An individual replica can maintain local tables that contain information that doesn't exist in the other replicas in the set.
- Share data among dispersed users. New information that is entered into the database while sales representatives are out of the office can be synchronized any time the sales representatives establish an electronic link with the corporate network. As part of their workday routine, sales representatives can dial into the network, synchronize the replica, and work on the most current version of the database. Because only the incremental changes are transmitted during synchronization, you can minimize the time and expense of keeping data up to date. Using *partial replicas*, it's also possible to synchronize only certain parts of the data.

For More Information For more information about partial replicas, see "Partial Replication" later in this chapter.

- Make server data more accessible. If your application doesn't need to have immediate updates to data, you can use database replication to reduce the network load on your primary server. Introducing a second server with its own copy of the database improves response time. You determine the schedule for synchronizing the replicas and can adjust that schedule to meet the changing needs of your users. Replication requires less centralized administration of the database while offering greater access to centralized data.
- Distribute application updates. When you replicate your application, you automatically replicate not only the data in your tables, but also your application's objects. If you make changes to the design of the database, Microsoft Jet transmits the changes during the next synchronization; you don't have to distribute new versions of the software.

- Guard against total data loss. Because replication distributes data across multiple databases, you can use one of the replicated databases to recover data if the original database is destroyed.

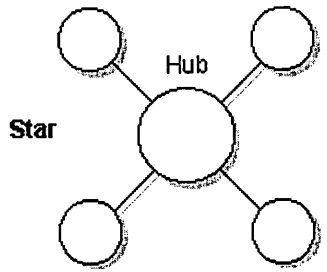
Although database replication can solve many of the problems inherent in distributed database processing, replication is less than ideal if:

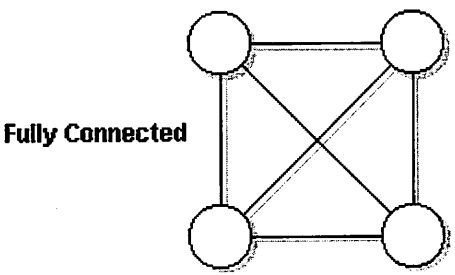
- You need to update a large number of records at multiple replicas. Applications that require frequent updates of existing records in different replicas are likely to have more record conflicts than applications that simply insert new records into a database. Record conflicts occur when users at different locations make any changes to the same record at the same time. Applications with more record conflicts require more administrative time because users must resolve the conflicts manually.
- Data consistency is critical. Applications that rely on information being correct at all times, such as fund transfers, airline reservations, and the tracking of package shipments, usually use a transaction method. While you can process transactions within a replica, there is no support for processing transactions across replicas. The information exchanged between replicas during synchronization is the *result* of the transaction, not the transaction itself.

Replica Set Topology

An important part of incorporating replication into your application is planning the appropriate topology for synchronizing data and design changes. A *replication topology* defines the communication between replicas and the logic that determines how synchronization occurs among the various replicas of the replica set.

Although you can implement any number of topologies, the star and fully connected topologies are usually the best for most applications.





In the star topology, a single hub periodically synchronizes with each of its satellite replicas. In this topology, all data is shared among replicas through a single, centralized database. The star topology reduces data latency time among replicas because data has to travel at most two "hops" to synchronize with any other replica.

If the average quantity of data to exchange during synchronization is relatively small, the star topology can be the most effective solution for most applications. It's reasonably simple to implement and can be both efficient and reliable. If you're implementing replication for the first time, the star topology is recommended.

In a fully connected topology, each replica synchronizes with all other replicas in the replica set. One way to conceptualize the fully connected topology is as a star topology where every replica acts as a hub. The fully connected topology has several benefits. One of the most significant benefits is the low latency time for data propagation. Because each replica synchronizes with every other replica, data is sent directly to all replicas without having to indirectly disperse through a series of replicas.

Replication with DAO

There is a Replication Manager application that allows the creation and maintenance of replica databases from within a standalone utility. This application ships with Microsoft Access, and is not available to Visual Basic Programmers unless you have also installed Microsoft Access. However, the Data Access Objects (DAO)

programming interface provides methods and properties to implement most replication features. (This is available through Microsoft Access 7.0 or later, or Microsoft Visual Basic 4.0 or later.) You can use DAO to:

- Convert a database into a Design Master.
- Make additional replicas.
- Create partial replicas.
- Synchronize replicas.
- Get and set specific properties of a replicated database.

Although DAO requires programming, it gives you the ability to customize your replication system. Generally, you should use DAO under the following circumstances:

- When you are a Visual Basic programmer who does not have access to the Replication Manager application that ships with Microsoft Access.
- When you need to synchronize replicas on a nonperiodic schedule or when certain events occur. For example, you might want to trigger synchronization whenever a replica receives updated product pricing information from headquarters. Because Replication Manager doesn't support event-based replication, you must use DAO to achieve this.
- When you need to distribute your replicated application to users who possess minimal computer expertise. In this case, you can use DAO to design a simplified replication interface or make replication completely transparent to users. Although you can distribute Replication Manager to your customers on a royalty-free basis, it might not be suitable for users with minimal knowledge of computer concepts.
- When you need to create a replica that contains only part of the data. For example, when you want to create a replica on a laptop that has minimal disk space.

Replicating a Database

You can use the following procedure to replicate your database with DAO.

▶ To replicate your database

1. Identify the objects in the database you don't want replicated and set their KeepLocal property to "T".
2. Make your database the Design Master by setting its Replicable property to "T".

   Alternatively, before setting the Replicable property, you can determine if the database is already replicated before you convert your database.
3. Make additional replicas from the Design Master by using the MakeReplica method.

Note Before using the KeepLocal and Replicable properties, you must create and append them to the Properties collection of the object that will use them. If you try to refer to these properties in your code before you create and append them, a run-time error occurs. The SetReplicable procedure, later in this chapter, shows how to do this.

Important The Microsoft Jet database engine doesn't allow you to protect a replicated database with a database password. Before you begin using replication, remove any database password protection from the database that you will be replicating. Setting user permissions doesn't interfere with replica synchronization.

Keeping Objects Local

When you convert a regular database into a Design Master, you replicate all of the objects in the database. If you don't want to disperse all objects in your database throughout the replica set, you can append and set the KeepLocal property on any objects you don't want to replicate. For example, if your database has a table containing confidential salary information, initialization information, or names of users who log on to the database, you might want to keep that information only at your replica. You can set the table's KeepLocal property to "T" to keep it local while all other objects are replicated when the database is replicated.

On TableDef and QueryDef objects, you create and append the KeepLocal property to the object's Properties collection. On forms, reports, macros, and modules defined by a host application, you create and append the KeepLocal property to the Properties collection of the Document object representing the object.

If you set the KeepLocal property and the property hasn't already been appended or inherited, an error occurs. The following code tries to create and append the KeepLocal property of a specified database object. If the property has already been appended, error 3367 occurs and the error handler sets the value of the property to "T":

```
Function SetKeepLocal(dbs As Database, strCollection _
   As String, strObject As String) As Integer

   Dim i As Integer
   Dim blnMatch As Boolean
   Dim tdf As TableDef
   Dim prp As Property
   Dim doc As Document
   Dim qdf As QueryDef

   On Error GoTo ErrorHandler
```

```
Select Case strCollection

      Case "Forms", "Reports", "Modules", "Scripts"
         Set doc = dbs.Containers(strCollection). _
            Documents(strObject)
         Set prp = doc.CreateProperty _
            ("KeepLocal", dbText, "T")
         doc.Properties.Append prp

      Case "TableDefs"
         Set tdf = dbs.TableDefs(strObject)
         Set prp = tdf.CreateProperty _
            ("KeepLocal", dbText, "T")
         tdf.Properties.Append prp

      Case "QueryDefs"
         Set qdf = dbs.QueryDefs(strObject)
         Set prp = qdf.CreateProperty _
            ("KeepLocal", dbText, "T")
         qdf.Properties.Append prp

   End Select

ErrorHandler:
Select Case Err
   Case 0
   Case 3265' Object not found.
      SetKeepLocal = ERR_OBJECTNOTFOUND
      Exit Function
   ' Property already exists, but make sure it is "T".
   Case 3367
      Select Case strCollection
         Case "Forms", "Reports", "Modules", "Scripts"
            doc.Properties("KeepLocal").Value = "T"
         Case "TableDefs"
            tdf.Properties("KeepLocal").Value = "T"
         Case "QueryDefs"
            qdf.Properties("KeepLocal").Value = "T"
            Exit Function
         End Select

      SetKeepLocal = SUCCESS
      Exit Function

   Case Else
      MsgBox "Error " & Err & ": " & Error, vbOKOnly
      SetKeepLocal = FAILED
      Exit Function
End Select

End Function
```

You cannot apply the KeepLocal property to objects that are already replicable.

If you have set a relationship between two tables in your database, you must set the KeepLocal property to the same value for both tables—both tables must be local or both must be replicable. If the property isn't set the same for both tables, the conversion will fail. However, you can't set the KeepLocal property while the relationship is in effect. Before setting the property, remove the relationship between the tables. After setting the KeepLocal property, add the relationship back to the two tables and proceed with converting the database.

Important If the object on which you are setting the KeepLocal property has already inherited that property from another object, the value set by the other object has no effect on the behavior of the object that you want to keep local. You must directly set the property for each object.

Replicating New Objects

If you create a new object in a replica and want to change it from local to replicable so that all users can use it, create the object in the Design Master, and set the object's Replicable property to "T". Be sure to delete the local object from any replicas; otherwise, an error occurs.

On TableDef and QueryDef objects, you create and append the Replicable property to the object's Properties collection, just as you do with the KeepLocal property. On forms, reports, macros, and modules defined by a host application, you create and append the Replicable property to the Properties collection of the Document object representing the object.

The next time you synchronize, the newly replicable object will appear in the other replicas in the set. Similarly, if there is a replicable object that you want to make local, set its Replicable property to "F".

Important If the object on which you are setting the Replicable property has already inherited that property from another object, the value set by the other object has no effect on the behavior of the object you want to replicate. You must directly set the property for each object.

Caution When you change the status of an object from replicable to local by setting the value of the object's Replicable property from "T" to "F", Microsoft Jet deletes the object from all replicas in the replica set except the Design Master. You should take extra care when you change the replicable status of a table. Even if you temporarily make a replicable table local at the Design Master and then make it replicable again, the next synchronization will delete and recreate the table at each replica. Unless you synchronize all replicas before making the design change, you will lose any data entered in the table at a replica since the last synchronization.

Making a Design Master

To make a database a Design Master, create the Replicable property for the Database object, append the property to the Database object's Properties collection, and set its Replicable property to "T".

The following code converts a database into a Design Master, and if the Replicable property doesn't exist, the code creates it and sets it to the specified value.

```
Public Function SetReplicable(strDB As String) _
   As Integer

   Dim prpReplicable As Property
   Dim dbsTarget As Database

   On Error GoTo ErrorHandler
   Set dbsTarget = OpenDatabase(strDB, True)

   ' Turn off error handling in case the
   ' property already exists.
   On Error Resume Next

   ' Create the Replicable property.
   Set prpReplicable = _
      dbsTarget.CreateProperty("Replicable", _
      dbText, "T")

   ' Append the property to the Properties collection.
   dbsTarget.Properties.Append prpReplicable

   ' Set the property to "T".
   dbsTarget.Properties("Replicable") = "T"
   SetReplicable = SUCCESS

   ErrorHandler:
   Select Case Err
      Case 0
         SetReplicable = SUCCESS
         Exit Function
      Case Else
         MsgBox "ERROR " & Err & ": " & Error
         SetReplicable = FAILED
         Exit Function
   End Select

End Function
```

During the conversion process, Microsoft Jet maintains all the property settings of your original database.

Note Microsoft Jet 3.5 also supports the Boolean ReplicableBool property. Its functionality is identical to the Replicable property.

Making Additional Replicas

Although you can make changes to the design of the database only in the Design Master, you can make additional replicas from any replica of the set. In fact, the only way to include new copies of the database in the replica set is to create them from an existing replica. Once created, all new replicas become part of the replica set. All the replicas have a unique identity and can communicate and synchronize with one another. Each replica set is independent from all other replica sets, and replicas in different sets cannot communicate or synchronize with each other.

Important Never try to make additional replicas from a copy of the original database before it was made the Design Master. The result wouldn't be an additional replica, but rather a new Design Master and replica set.

When you convert your database into a Design Master by setting its Replicable property to "T", you have only one replica (the Design Master) in the replica set, and you make your first replica from it. You can make your first replica, and subsequent replicas, with the MakeReplica method.

The following code makes an additional replica.

```
Function MakeAdditionalReplica(strReplicableDB As _
   String, strNewReplica As String, intOptions As _
   Integer) As Integer

   Dim dbs As Database

   On Error Goto ErrorHandler

   Set dbs = OpenDatabase(strReplicableDB)

   ' If no options are passed to
   ' MakeAdditionalReplica, omit the
   ' options argument, which defaults to
   ' a full, read/write replica. Otherwise,
   ' use the value of intOptions.

   If intOptions = 0 Then
      dbs.MakeReplica strNewReplica, _
         "Replica of " & strReplicableDB
   Else
      dbs.MakeReplica strNewReplica, _
         "Replica of " & strReplicableDB, _
         intOptions
   End If
   dbs.Close
```

```
ErrorHandler:
   Select Case Err
      Case 0:
         MakeAdditionalReplica = SUCCESS
         Exit Function
      Case Else:
         MsgBox "Error " & Err & " : " & Error
         MakeAdditionalReplica = FAILED
         Exit Function
   End Select

End Function
```

The `intOptions` argument can be a combination of the DAO constants dbRepMakeReadOnly and dbRepMakePartial, or it can be 0. The dbRepMakeReadOnly constant indicates that the replica is read-only. The dbRepMakePartial constant indicates that the replica is a partial replica rather than a full replica.

For example, to create a read-only partial replica, the call to MakeAdditionalReplica should pass the value `dbRepMakeReadOnly + dbRepMakePartial` as the value of `intOptions`.

If `intOptions` is 0, the function creates a full, read/write replica because the *options* argument of the MakeReplica method is omitted.

For More Information For more information about partial replicas, see "Partial Replication" later in this chapter.

When Microsoft Jet creates the new replica, the new replica will contain all data definition language property settings of the source replica. You can make subsequent replicas from either the Design Master or any replica in the set.

When you use the MakeReplica method, be sure that the objects you are replicating aren't locked, or the method will fail. Microsoft Jet locks objects while they are open in design mode or being updated during a synchronization. Programmers might easily overlook this requirement and attempt to make a replica from the database that has locked objects.

Note When you make a new replica, you copy all of the replicable objects and properties from the source replica to the new replica. Although you copy all linked tables, the path to a linked table may no longer be accurate because of the new replica's location on the network. Be sure to test the new replica to determine whether you need to establish a new path for any of the linked tables.

Changes to Your Database

When you replicate your database, Microsoft Jet:

- Adds new properties to your database.
- Adds new fields to each existing table in your database.
- Adds new tables to your database.
- Increases the physical size of your database file as a result of the additional fields and tables.
- Changes the behavior of AutoNumber fields.

Microsoft Jet uses these changes to track modifications to your database's design and data and to efficiently synchronize your database with other replicas in the replica set.

Replication Properties

Microsoft Jet adds the following properties to replicated databases: Replicable, ReplicaID, and DesignMasterID (for Design Masters only).

The Replicable property is set to "T", indicating the database is a replica. However, after the property is set to "T", you cannot change it. If you set the property to "F" (or any value other than "T"), an error occurs.

The ReplicaID property is a 16-byte value that provides the Design Master or replica with a unique identification.

The DesignMasterID property identifies the Design Master in the replica set. You should set this property only at the current Design Master. Under extreme circumstances—for example, the loss of the original Design Master—you can set the property at the current replica. However, setting this property at a replica when there is already another Design Master might partition your replica set into two irreconcilable sets and prevent any further synchronization of data. If you determine that it's necessary to set this property at a replica, first synchronize the replica with all other replicas in the set.

Caution Never create a second Design Master in a replica set. The existence of a second Design Master can result in the loss of data.

New Fields

The conversion process adds several new system fields to each existing table in a database. Among these fields are a unique identifier, a generation indicator, and a lineage indicator. You will only see the unique identifier and generation indicator fields.

Unique Identifier

During the conversion of a database, Microsoft Jet first looks at the existing fields in a table to determine whether any field uses both the AutoNumber data type and Replication ID field size. If no field uses that data type and field size, Microsoft Jet adds the s_Guid field, which stores the Replication ID AutoNumber that uniquely identifies each record. The Replication ID AutoNumber for a specific record is identical across all replicas.

The Replication ID AutoNumber is a 16-byte GUID (Globally Unique Identifier, sometimes referred to as a UUID, or Universally Unique Identifier) that appears in the following format.

{2FAC1234-31F8-11B4-A222-08002B34C003}

The hyphens and braces are used in display only and don't form part of the GUID.

There can be more than one field in a table of type GUID, but there can be only one autogenerated field of type GUID in a table (regardless of whether it's replicable). If an autogenerated GUID field is added to a table, GUID values are generated for all records in the table. You cannot change or delete the value of an autogenerated GUID.

Generation Identifier

During the conversion process, Microsoft Jet also adds a field called s_Generation to each table in the database. The s_Generation field helps expedite incremental synchronization by preventing a replica from sending records that haven't been updated since the last synchronization. Whenever a record changes, its generation is set to 0.

Usually during a synchronization all records with generation 0 are sent to the other replicas, and then the generation for the record is incremented to one more than the last generation, which now becomes the new highest generation. The sending replica knows the last generation sent to that specific receiving replica, and sends only records with generation 0 and generations higher than the previous generations.

The receiving replica doesn't apply generations received out of sequence. In some cases, Microsoft Jet replication might determine that there are too many records to send in a single message.

Generally, there is a single generation field per record. To optimize synchronizations for databases that contain Memo or Long Binary fields (sometimes referred to as a BLOBs, or binary large objects), an extra generation field is associated with each BLOB. If the BLOB is modified, this generation value is set to 0 so that the BLOB is sent during the next synchronization. If a record modifies other fields, but not the BLOB, the BLOB generation isn't set to 0 and the BLOB isn't sent.

This extra generation field is named Gen_*CommentFieldName*, where *CommentFieldName* is the comment field name (truncated, if necessary). If this name duplicates an existing field name, the prefix changes, one character at a time, until a unique name is found. This sets the ColGeneration property for the comment field to identify the new Gen_*CommentFieldName* field. One of these fields is set for each comment field.

Replication System Tables

Microsoft Jet adds several system tables to the database during the conversion process. Most of these tables are system tables, which aren't usually visible to users and cannot be manipulated in code. The following table describes the most important tables.

Name	Description
MsysRepInfo	Stores information relevant to the entire replica set, including the identity of the Design Master. It contains a single record. This table appears in all replicas in the replica set.
MsysReplicas	Stores information about all replicas in the replica set. This table appears in all replicas in the replica set.
MsysTableGuids	Relates table names to GUIDs. TableGuids are used in tables such as MSysTombstone as a reference to a table name stored in this table. This allows efficient renaming of tables. In addition, this table includes the level number used for ordering tables so that updates can be processed efficiently. This is a local table that is updated by the tracking layer at the Design Master and, as part of the processing of design changes, at all other replicas in the replica set.
MsysSchemaProb	Identifies errors that occurred while synchronizing the design of the replica. This is a local table that exists only if a design conflict has occurred between the user's replica and another replica in the replica set.
MsysErrors	Identifies where and why errors occurred during data synchronization. This table appears in all replicas in the replica set.
MsysExchangeLog	Stores information about synchronizations that have taken place between this replica and other replicas in the replica set. This is a local table.
MsysSidetables	Identifies the tables that experienced a conflict and the name of the conflict table that contains the conflicting records. This is a local table that exists only if a conflict occurs between the user's replica and another replica of the set.
MsysSchChange	Stores design changes that occurred at the Design Master so that they can be dispersed to any other replica in the replica set. The records in this table are deleted periodically to minimize the size of the table.

Name	Description
MsysTombstone	Stores information on deleted records, and allows deletions to be dispersed to other replicas. This table appears in all replicas in the replica set.
MsysTranspAddress	Stores address information for Synchronizers and defines the set of Synchronizers known to this replica set. This table appears in all replicas in the replica set.
MsysSchedule	Stores information for scheduled synchronization. The Synchronizer for a local replica uses this table to determine when the next synchronization with another Synchronizer should take place, and how to synchronize data and design changes with the other Synchronizer.
MsysGenHistory	Stores a history of generations. It contains a record for each generation that a replica knows about. This prevents Microsoft Jet from sending common generations during synchronizations and resynchronizing replicas that are restored from backups. This table appears in all replicas in the replica set, but it's merged by a process slightly different from that used with normal replicated tables.
MsysOthersHistory	Stores a record of generations received from other replicas. It contains one generation from every message sent from other replicas.

Increases in the Physical Size of Your Database

The addition of three new fields during the conversion process imposes two limitations on your tables. First, Microsoft Jet allows a maximum of 2048 bytes (not counting Memo or Long Binary fields) in a record. Replication uses a minimum of 28 bytes to store unique identifiers and information about changes to the record. If the record contains either Memo or Long Binary fields, replication uses an additional 4 bytes for each of those fields. You can calculate the total number of bytes available in a record in a replicated table as follows:

2048 bytes
−28 bytes for replication overhead
−(4 bytes * the number of Memo fields)
−(4 bytes * the number of Long Binary fields)
= the maximum number of bytes available

Second, Microsoft Jet allows a maximum of 255 fields in a table, of which at least 3 fields are used by replication. You can calculate the total number of fields available in a replicated table as follows:

255 fields
−3 system fields
−the number of Memo and Long Binary fields
= the number of fields available

Few well-designed applications use all the available fields in a table or characters in a record. However, if you have a large number of Memo or Long Binary fields in your table, you should be aware of your remaining resources.

In addition to decreasing the available number of characters and fields, Microsoft Jet replication also imposes a limitation on the number of nested transactions. You can have a maximum of seven nested transactions in a regular database, but a replicated database can have a maximum of six nested transactions.

Just as the addition of three new fields to your tables adds to the size of each record, the addition of new system tables adds to the size of your database. Many of these new tables contain only a few records, but some of the new tables can grow significantly depending upon the frequency of synchronization between replicas.

The size of your database file is significant for three reasons:

- Microsoft Jet supports files up to a maximum of 1 GB in size.
- Many users have limited space available on their hard disks, and the creation of a replicated database larger than the original file can use up all available disk space.
- If you choose to make a backup copy of your replicated database, but as a regular database, you must have sufficient space on your hard disk.

Changes to the Behavior of AutoNumber Fields

When you make a regular database a replicated database, the AutoNumber fields in your tables change from incremental to random numbering. All existing AutoNumber fields in existing records retain their values, but new values for inserted records are random numbers. The record numbers don't reflect the order in which the records were inserted, and consequently, the record inserted last doesn't necessarily have the highest value.

When you open a table with a random AutoNumber key, the records appear in the order of ascending random numbers, not in chronological order. With random AutoNumber fields it's possible—although highly unlikely—for records inserted at different replicas to be assigned the same value. If this happens, updates could take place in the wrong records. If you experience problems such as these, consider using the s_Guid field as the primary key.

For More Information For more information about using the s_Guid field as the primary key, see "Using Replication IDs as Primary Keys" later in this chapter.

Before you make a regular database a replicated database, determine whether any of your applications or users rely on the order and incremental nature of the AutoNumber field. If so, you can use an additional Date/Time field to provide sequential ordering information.

Database Security

Replicated databases use the same security model as regular databases; users' permissions on the database are determined at the time they start Microsoft Access and log on. As a programmer, you should make sure that the same security information is available at each replica. You can do this by making the identical workgroup information file (the file that stores security information) available to users at each replica location. The default security workgroup information file is called System.mdw.

Security permissions control certain aspects of database replication. For example, a user must have Administer permission on the database to:

- Make a regular database a Design Master.
- Make a local object replicable or make a replicable object local.
- Designate a replica as the new Design Master. You can perform this action in any replica in the set; however, you should never have more than one Design Master in a set at one time.

By default, the Users group, the Admins group, and the creator of the database have Administer permission. If you need to maintain security, you must restrict this permission to selected users.

Take care to ensure that there is always at least one user with the Administer permission on the database. It's possible that a site with the Design Master and its associated System.mdw could be destroyed through a hard disk failure. While it's possible to use another replica to retain access to the data, you will need to make one of the replicas the Design Master. Only a user with Administer permission on the database can do this. Therefore, you must make sure the System.mdw at the remote site provides Administer permission on the database to one of the users.

Synchronizing Replicas

For database replication to be useful, replicas must communicate with one another to remain up to date. Synchronization is the process of making the design and data in all the replicas identical. Data changes in one replica are periodically communicated to each of the other replicas.

You can synchronize one replica with another replica in the replica set by using DAO.

To synchronize replicas with DAO code, use the Synchronize method. You can also perform one-way or two-way exchanges. For example, the following code fragment uses dbRepImpExpChanges to perform a two-way exchange between replicas:

```
Sub SynchronizeDB(dbs As Database, _
   strSyncTargetDB As String)

   ' Synchronize replicas (bidirectional exchange).
   dbs.Synchronize strSyncTargetDB, dbRepImpExpChanges
   dbs.Close

End Sub
```

If you don't provide an *exchange* argument, the synchronization is bidirectional (import and export).

Microsoft Jet always synchronizes design changes before it synchronizes data changes. The design of both replicas must be at the same version level before data can be synchronized. For example, even if you use dbRepExportChanges to specify that data changes flow only from the current replicas to the designated target replica, a user could make design changes to the current replica if it has a lower version number than the target replica.

If you want to prevent users from making changes to the design of your database, don't put the Design Master on the network server. Instead, keep the Design Master at a network location that is accessible only to you. When you make changes to your application, you can synchronize with the replica on the server and rely on it to pass these changes on to other replicas in the replica set.

Synchronizing the Star Topology

As described in "Replica Set Topology" earlier in this chapter, you can implement a variety of topologies for synchronizing replicas in the replica set.

The following code example shows one way of implementing the star topology. The table specified in `strReplicas` stores a list of replica paths. The database specified in `strDB` contains the `strReplicas` table. The table must have a field named Path that contains the replica paths. The name of the replica acting as the hub is specified in `strHub`. If failures occur during synchronization, the replica at which the failure occurred is stored in the string array `strFailures`, which is available to the calling statement.

```
Function StarSync(strDB As String, strFailures() As _
   String, strReplicas As String, strHub As String) _
   As Integer
```

```
Dim dbsHub As Database
Dim dbsPaths As Database
Dim rstReplicas As Recordset
Dim i as integer

On Error GoTo ErrorHandler

Set dbsPaths = OpenDatabase(strDB)
Set rstReplicas = dbsPaths.OpenRecordset _
   (strReplicas, dbOpenSnapshot)

On Error Resume Next
Set dbsHub = OpenDatabase(strHub)

' If the hub replica specified in strHub is
' unavailable, return an application error so the
' caller can reassign a new hub replica and call
' the function again.
If Err <> 0 Then
   StarSync = FAILED
   Exit Function
End If

' Synchronize all replicas in the replica set
' with the hub. Because error trapping is turned
' off, the program continues to synchronize with
' other replicas if the Synchronize method fails
' for any replica.

Err = 0
Do Until rstReplicas.EOF
   If rstReplicas!Path <> strHub Then
      ' The hub can't synchronize with itself.
      dbsHub.Synchronize rstReplicas!Path
      End If

   ' If an error occurred while synchronizing with
   ' the current replica, log an error that stores
   ' which replica failed to synchronize.

   If Err <> 0 Then
      ReDim Preserve strFailures(1 To _
         Ubound(strFailures) + 1)
      strFailures(i) = rstReplicas!Path
      Err = 0
   End If

   rstReplicas.MoveNext
Loop

dbsHub.Close
rstReplicas.Close
dbsPaths.Close
Exit Function
```

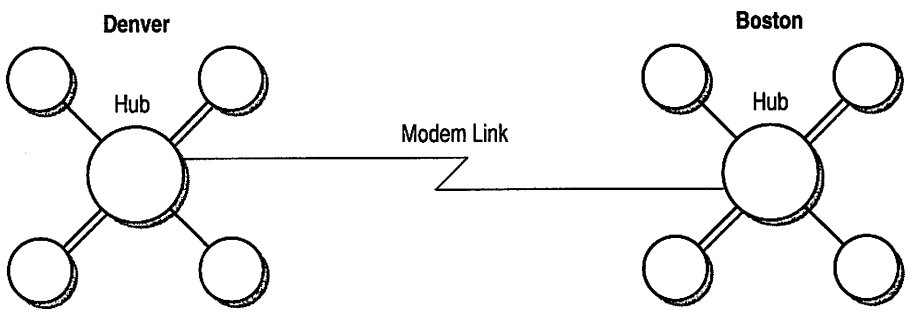
```
ErrorHandler: ' Error handler
   dbsHub.Close
   rstReplicas.Close
   dbsPaths.Close
   StarSync = Err
   Exit Function

End Function
```

Synchronizing Data Over Wide Area Networks

Synchronization strategies for computers connected by a single local area network (LAN) are usually quite different from synchronization strategies for computers that are connected using wide area networks (WANs) and modems. For computers connected through a single LAN, synchronization can occur frequently because telecommunication overhead isn't an issue.

With WANs, telecommunication overhead can be a major factor because of longer data transmission times and higher transmission rates. This section discusses two different WAN topologies along with possible ways to handle synchronization.

The following diagram shows a company that has an order entry department in Denver and headquarters in Boston. Each day, customers call an 800 number to order products from the company's catalog, and order takers can enter these orders into a custom Microsoft Jet database application.



Once a day, sales transactions entered in Denver must be transferred to headquarters, where managers can run queries against the data to track sales totals, geographic sales distribution, and other statistics. Conversely, updated product information, such as new products and discounts on existing products, must be sent from headquarters to the sales office as soon as it is available. Computers in the order entry department comprise a single LAN, and computers at headquarters comprise another LAN. The two LANs are connected through a modem.

In this environment, how can replication meet the company's requirements? Configuring replication within each LAN is a fairly straightforward task. For example, the replicas in Denver could synchronize data every 30 minutes using a star topology. This synchronization occurs repeatedly at specified intervals set in code using DAO methods and a form's Timer event. The replicas in Boston can synchronize data in a similar manner.

The main concern is the bottleneck created by the modem link between Denver and Boston. For a large number of sales transactions, the telecommunication overhead would be high due to the frequency of synchronizing data between the two cities. However, because headquarters needs information only once a day, a good strategy would be to wait until night to synchronize one of the computers in Denver with one of the computers in Boston. The computer in Boston and the computer in Denver must be fully synchronized within their respective stars before they synchronize between the two stars. Synchronizing the two star hubs will then synchronize the data in the two cities.

Because product information in Boston must be sent to Denver only when new information is available, replication can occur only when necessary. This may occur once or twice a week. To handle this, a user can click a button on a form to synchronize the databases in Denver and Boston whenever product information changes. A more sophisticated solution is to set a value in a table each time information in one of the product tables changes. A procedure linked to a Timer event periodically checks this value and invokes the Synchronize method whenever the value is set.

Note Networks have topologies too, and these shouldn't be confused with replication topologies. Although the diagram shows a star topology, this refers to the replication topology. The illustration doesn't show an underlying network topology, which can be of almost any type.

Synchronizing Data Over the Internet

In DAO code, synchronizing with a replica on an Internet server is similar to synchronizing with a replica on a LAN or WAN. Instead of specifying a network path to the server, you specify a Uniform Resource Locator (URL) address to identify resources on the Internet. The following code fragment shows how to synchronize a database with a replica on an Internet server. In this case, the URL of the Internet server is http://www.mycompany.myserver.com:

```
Dim dbs As Database
Set dbs = OpenDatabase("C:\Data\OrdEntry.mdb")
' Synchronize the local database with the replica on
' the Internet server.
dbs.Synchronize "http://www.mycompany.myserver.com" _
   & "/files/Orders.mdb", dbRepImpExpChanges + _
   dbRepSyncInternet
dbs.Close
```

When processing the Synchronize method, Microsoft Jet looks for the presence of the dbRepSyncInternet constant and takes the necessary steps to connect with the Internet server at the specified URL address. Microsoft Jet automatically performs all the necessary communication operations to establish the server connection.

Note You can't synchronize a full replica with a partial replica over the Internet. You can only synchronize two full replicas over the Internet.

For More Information For more information about partial replicas, see "Partial Replication" later in this chapter.

As with synchronizing over a LAN or WAN, you can set the direction in which data is replicated over the Internet by using the *exchange* argument of the Synchronize method. In the preceding example, dbRepImpExpChanges indicates that changes in both the client replica and the Internet server replica are replicated. To replicate changes in only one direction, set the *exchange* argument to either dbRepImportChanges or dbRepExportChanges.

How Internet Replication Works

Because it isn't possible to perform a direct synchronization between two databases over the Internet (as you can on a LAN), the File Transfer Protocol (FTP) folder on the Internet server acts as a dropbox folder for indirect synchronizations.

When a client invokes a Synchronize method to synchronize changes with an Internet server, several dynamic-link libraries (DLLs) on the client and server computers work together to perform the synchronization.

First, the client connects to an HTTP address with the Internet server, and the server receives a command to start the Synchronizer. This command includes a command-line argument indicating which replica requested the synchronization.

The client then builds a message file containing all the design and data updates that occurred since the last synchronization and sends it to the FTP folder on the Internet server. The Synchronizer on the Internet server then applies these changes to its local replica. Next, the Synchronizer builds a message file containing all the design and data updates that occurred since it last synchronized with the client replica, and it places this file in the server's FTP folder. The server sends the name and location of the message file back to the client workstation, which transfers the message file from the Internet server to the client using FTP. All the changes specified within the message file are then applied to the client replica, and the synchronization is complete.

Note It isn't necessary to have a Synchronizer installed on the client workstation to synchronize databases over the Internet. The Microsoft Jet and Microsoft Windows DLLs handle all aspects of the synchronization. Depending on how the client workstation and Internet server are configured, the establishment of a connection with the server may result in the display of a logon dialog at the client workstation. In this case, the synchronization won't occur unless a user at the client workstation types the appropriate user name and password into the logon dialog.

Synchronizing Multiple Replicas

In some cases, you may have to synchronize a group of computers at the same location with a replica on an Internet server. For example, all computers in a company's marketing department may need to synchronize with a replica over the Internet to update a sales forecasting application to the latest version.

When synchronizing multiple replicas with a replica on the Internet, you might want to dedicate a single replica to synchronize with the replica on the Internet server. All the other replicas in the department can then synchronize with the dedicated replica over a local area network. This is usually much more efficient than having all replicas synchronize individually with the Internet server.

Important When creating a replicated database system, be aware that there is a limit of approximately 64,000 replicas that can synchronize with the replica on the Internet server. Replication over the Internet is therefore designed mainly for limited numbers of users in a controlled situation. It isn't intended to be a system that can be accessed by an arbitrary numbers of users, as is the case with a Web page.

Partial Replication

So far, this chapter has discussed *full replication*, in which all records in all replicas of a replica set are synchronized in their entirety. But what if you don't want to replicate all of the data? For example, a business might store its entire sales database at the headquarters office but only replicate regional data to its regional offices across the country. To do this, you create *partial replicas*, which contain only a subset of the records in the full replica. With a partial replica, you can set filters and relations that identify which subset of the records in the full replica you want to transfer.

To create a partial replica, you must use DAO objects. Once you've created a partial replica, you can control partial replication using two properties:

- ReplicaFilter is a property of the TableDef object that you can use to specify which subset of records is replicated to the table from the full replica. This property uses syntax similar to an SQL WHERE clause.

- PartialReplica is a property of the Relation object that you can use to specify which relationships you will use to force the inclusion of related records. This enables you to replicate data from the full replica to the partial replica based on relationships between tables.

Note You can synchronize data only between a full replica and a partial replica. You can't synchronize data between two partial replicas. Also, with partial replication, you can set restrictions on which records are replicated, but you can't indicate which fields are replicated.

Advantages of Partial Replication

Partial replication is useful for a number of purposes. For example, you can use it to help restrict access to data. In the case of a sales database, partial replication can help ensure that people in a regional sales office won't view sales data for other regional offices. Although you can use partial replication to restrict access to records, it isn't a substitute for a security system.

Partial replication also has benefits for replicating data over LANs and WANs. By applying filters and relations that restrict which data is replicated, partial replication can reduce the amount of data transferred over a local area network or a long-distance carrier. This can reduce network traffic and lower telecommunications costs.

Working with Partial Replicas

To write DAO code that uses partial replication, you'll need to understand how to create a partial replica, how to set a replica filter, and how to set the table relationships that apply to replication.

Creating Partial Replicas

You can use DAO methods and properties to create a partial replica.

▶ **To create a partial replica**

1. Use the MakeReplica method on a Database object, specifying the dbRepMakePartial constant in the *options* argument. This creates the partial replica. The MakeAdditionalReplica function in "Making Additional Replicas" demonstrates how to do this.
2. Use the ReplicaFilter and PartialReplica properties to set the desired filters and relationships that determine which data to replicate from the full replica.
3. Call the PopulatePartial method to transfer all records from the full replica that meet the new replica filter criteria.

Note Once you create a partial replica, you cannot convert it to a full replica. However, if you remove all of the replica filters and replica relations within the partial replica, it will contain all the records of a full replica, but it will still have the same limitations and restrictions of a partial replica.

Setting Replica Filters

The replica filter specifies which records will be included in a replica. The ReplicaFilter property can take the following values.

Value	Description
A string	A criteria that a record must satisfy to appear in the replicated table. It's similar to an SQL WHERE clause, but you cannot specify subqueries, aggregate functions (such as Count), or user-defined functions within the criteria.
True	Replicate all records.
False	(Default) Don't replicate any records.

For example, to replicate only customer records from the California region, your code might look like the following:

```
Sub ReplicaFilterX(strPartialReplica As String)

   Dim tdfCustomers As TableDef
   Dim strFilter As String
   Dim dbs As Database

   Set dbs = OpenDatabase(strPartialReplica)

   Set tdfCustomers = dbs.TableDefs("Customers")
   strFilter = "Region = 'CA'"

   tdfCustomers.ReplicaFilter = strFilter

   ' ...Insert code to synchronize records...

   ' Now remove the replica filter (for example
   ' purposes only).
   tdfCustomers.ReplicaFilter = False
   dbs.Close

End Sub
```

As the example shows, to remove a filter, set the ReplicaFilter property to False. Once you remove all filters and invoke the PopulatePartial method, no records will appear in any replicated tables.

Usually, you reset a replica filter when you want to replicate a different set of records.

For example, when a sales representative temporarily takes over another sales representative's region, the database application can temporarily replicate data for both regions and then return to the previous filter. In this scenario, the application resets the ReplicaFilter property and then repopulates the databases.

Tip You can significantly improve synchronization speed by specifying indexed fields in the ReplicaFilter property.

The Role of Referential Integrity

If your database contains relationships that enforce referential integrity, related records are automatically replicated as needed. For example, suppose you create a new replica with no records in it and set the ReplicaFilter property of the Orders table in the new replica to `"Amount > 1000"`. The next synchronization will populate the Orders table in the new replica with all records with an order amount over $1,000. This will also populate records in the Customers table that are related to the Orders records, along with records in the Products table that are related to the records in the Orders table, and so on.

This automatic inclusion only occurs for records that must satisfy referential integrity constraints. For example, because the Orders table has a many-to-one relationship to the Customers table, if one or more records exist in the Orders table for a given customer, the Customers table must have a corresponding record in order to satisfy the referential integrity rule. However, placing a ReplicaFilter property on the Customers table doesn't automatically force the inclusion of related records in the Orders table. This is because referential integrity rules don't require a referenced row. To include such rows, use the PartialReplica property described in the following section.

Setting Replica Relations

In some cases, setting the ReplicaFilter property alone can't adequately specify what should happen during synchronization. For example, suppose you have a database in which the Customers table has a one-to-many relationship with the Orders table, and you want to configure a partial replica that only replicates orders from customers in the California region (instead of all orders). You can't set the ReplicaFilter property on the Orders table to `Region = 'CA'` because the Region field is in the Customers table, not the Orders table.

To replicate all orders from the California region, you must indicate that the relation between the Orders and Customers tables will be active during replication. Therefore, once you've created a partial replica, you'll need to perform two steps to retrieve all orders from the California region:

- Set the ReplicaFilter property on the Customers TableDef object to `"Region = 'CA'"`.
- Set the value of the PartialReplica property to True on the Relation object corresponding to the relationship between Orders and Customers.

The following code fragment shows how to set the PartialReplica property:

```
' Assumptions:
' The current open database, dbs, is the partial
' replica, and appropriate relationships already exist
' between the Orders and Customers tables.

' Find the "Customers to Orders" Relation object.
For intI = 0 To dbs.Relations.Count - 1
   If (dbs.Relations(intI).Table = "Customers") _
      And (dbs.Relations(intI).ForeignTable = _
      "Orders") Then
      ' Set the Relation object's PartialReplica
      ' property to True.
      dbs.Relations(intI).PartialReplica = True
      Exit For
   End If
Next intI
```

Now suppose you have a second requirement to replicate all orders greater than $1000, regardless of the region with which they are associated. To do this, you can set the ReplicaFilter property of the Orders table to `"Amount > 1000"`. Synchronizing the partial replica with the full replica will populate the Orders table with all orders greater than $1000 from all regions, in addition to all orders from the California region.

Notes

- If you have set a replica filter and a replica relation on the same table, the two act in combination as a logical Or operation, not a logical And operation. For instance, in the preceding example, the records exchanged during synchronization are all orders greater than $1000 or all orders from the California region, not all orders from the California region that are over $1000.
- It makes no difference in which order you add replica filters and replica relations.
- Although it's possible to set complex filters and replica relationships, this isn't recommended. You should use the ReplicaFilter and PartialReplica properties to place general restrictions on which data is replicated. Don't use them the same way that you would use a query.

Repopulating Partial Replicas

When you synchronize a partial replica with a full replica, it is possible to create "orphaned" records in the partial replica. For example, suppose you have a Customers table with its ReplicaFilter set to `"Region = 'CA'"`. If a user changes a customer's region from CA to NY in the partial replica, and then a synchronization occurs via the Synchronize method, the change is propagated to the full replica but the record containing NY in the partial replica is orphaned because it now doesn't meet the replica filter criteria.

To solve the problem of orphaned records, you can use the PopulatePartial method. The PopulatePartial method is similar to the Synchronize method, but it pushes any unpropagated changes to the full replica, removes all records in the partial replica, and then repopulates the partial replica based on the current replica filters. Even if your replica filters have not changed, PopulatePartial will always repopulate the partial replica by clearing all records in the partial replica.

Generally, you should use the PopulatePartial method when you create a partial replica and whenever you change your replica filters. If your application changes replica filters, you should follow these steps:

1. Synchronize your full replica with the partial replica in which the filters are being changed.
2. Use the ReplicaFilter and PartialReplica properties to make the desired changes to the replica filters.
3. Call the PopulatePartial method to remove all records from the partial replica and transfer all records from the full replica that meet the new replica filter criteria.

If a replica filter has changed, and the Synchronize method is invoked without first invoking PopulatePartial, a trappable error occurs.

Note Although PopulatePartial performs a one-way synchronization before clearing and repopulating the partial replica, it is still a good idea to call Synchronize before calling PopulatePartial. This is because if the call to Synchronize fails, a trappable error occurs. You can use this error to decide whether or not to proceed with the PopulatePartial method (which removes all records in the partial replica). If PopulatePartial is called by itself and an error occurs while records are being synchronized, records in the partial replica will still be cleared in the partial replica, which may not be the desired result.

Handling Conflicts and Errors

When using database replication, you may occasionally encounter synchronization conflicts, synchronization errors, or design errors. *Synchronization conflicts* occur when users update the same record at two replicas in the replica set and Microsoft Jet attempts to synchronize the two versions. The synchronization succeeds, but the changes at only one of the replicas are applied to the other. *Synchronization errors* occur when a data change at one replica cannot be applied to another replica because it would violate a constraint, such as a referential integrity rule or uniqueness assertion. The synchronization succeeds, but the content of replicas is different. *Design errors* occur when a design change at the Design Master conflicts with a design change at a replica. The synchronization fails and the content of different replicas starts to diverge.

Synchronization errors and design errors are more significant problems than synchronization conflicts because the replicas no longer share a common design or identical data. This section describes the factors that contribute to conflicts and errors, and suggests ways to prevent or resolve them.

Synchronization Conflicts

Whenever you synchronize two replicas, there is always the possibility of conflict between versions because the same record might have changed at both locations. During synchronization, if Microsoft Jet detects that the same record changed at both replicas, Microsoft Jet treats it as a synchronization conflict. Even if the changes were in the same record but in different fields, Microsoft Jet treats it as a synchronization conflict.

When a synchronization conflict occurs, Microsoft Jet doesn't attempt to resolve the conflict based on the content of the records or the data changes. Instead, it uses a simple algorithm to select one of the record versions to be the official change, and it writes the data from the other record version into a conflict table. It is then the responsibility of the programmer, database administrator, user, or a custom resolution program to review each conflict to determine that the correct information was applied to the database.

The algorithm Microsoft Jet uses to select one record version over the other is based on the record's version number, which is stored in the table's s_Lineage field. Each time data changes in a record, the version number increases by one. For example, a record with no changes has a version number of 0. A change to data increases the version number to 1. A second change to the same data, or a change to different data in the record, changes the version number to 2. If the record at one replica has changed once and the same record at the second replica has changed three times, then the record at the second replica has a higher version number than the record at the first replica. When Microsoft Jet compares the version numbers for the same record, it assumes that the version that changed the most is the more correct of the two versions. If both records have the same version number, Microsoft Jet examines the ReplicaID and selects the replica with the lowest ReplicaID.

Even though this algorithm and its underlying assumption might not be the most appropriate in all applications, it allows Microsoft Jet to quickly and consistently complete the synchronization process and preserves all the information from both versions for later review. The data from the record version that Microsoft Jet didn't select is stored in a conflict table in the replica where the change was made. Conflict tables derive both their names and fields from the underlying tables. Conflict table names are in the form *table*_Conflict, where *table* is the original table name. For example, if the original table name is Customers, the conflict table name is Customers_Conflict. Because conflicts are reported only to the replica that originated the losing update, conflict tables aren't replicated.

Resolving Conflicts Using DAO

After synchronizing two replicas, you should review each conflict to determine that the database contains the correct information. You can determine whether a conflict occurred for a specific table by using the ConflictTable property. This property returns the name of the conflict table containing the database records that conflicted during synchronization.

You can use the following code to find the name of the conflict table, examine each record conflict, and take action to resolve each conflict.

```
Sub Resolve(dbs As Database)

   Dim tdfTest As TableDef
   Dim rstConflict As Recordset

   For Each tdfTest In dbs.TableDefs
   ' If a conflict table exists then open the
   ' conflict table.
   If (tdfTest.ConflictTable <> "") Then
      Set rstConflict = _
         dbs.OpenRecordset(tdfTest.ConflictTable)
      ' Process each record.
      rstConflict.MoveFirst
      While Not rstConflict.EOF
         ' ...Insert code here to resolve conflicts...
         ' Remove conflicting record when finished.
         rstConflict.Delete
         rstConflict.MoveNext
      Wend
      rstConflict.Close
   End If
   Next tdfTest

End Sub
```

If there is no conflict table, or if the database isn't a replica, the property returns a zero-length string.

As you review each conflict in the conflict table, you should take the appropriate action. If Microsoft Jet selects the correct record version and no further action is necessary, you can delete the record from the conflict table. If Microsoft Jet selects the wrong record version, you might want to:

- Manually enter the data from the conflict table into the database.
- Develop a custom conflict-resolution routine that always assigns a higher priority to changes in one specific replica over another replica.
- Consider whether procedural changes regarding how data is entered or changed in records are necessary.

Synchronization Errors

You need to be aware of at least four sources of potential synchronization errors when building your application:

- Table-level validation (TLV) rules
- Duplicate keys
- Referential integrity
- Record locks

Table-Level Validation Rules

Microsoft Jet allows you to establish table-level validation (TLV) rules to restrict the value or type of data entered into a table. However, if you implement TLV without determining that any existing data doesn't violate the validation rule, you might encounter a synchronization error in the future.

For example, assume that you apply a TLV rule to a field that already contains data, and you test it to make sure that the existing values meet the rule. There might be new or updated records at other replicas that don't satisfy the new rule. If you don't test the existing values at the other replicas and change the values that fail the rule, the next synchronization will fail to correctly update the records. When Microsoft Jet applies the new or updated records from the other replica to the replica on which you set the TLV rule, Microsoft Jet tests each insertion or update against the rule. If the value fails, the update fails, and Microsoft Jet records an error in the MSysErrors table at the receiving replica. To correct the error, you must correct the invalid values at the sending replica. You can avoid the error by synchronizing all replicas in the replica set before applying a TLV rule.

Duplicate Keys

Duplicate keys can occur when two users at different replicas either simultaneously insert a new record and use the same primary key for their respective records or change the key on two different records when both happen to use the same value. When the replicas are synchronized, the synchronization succeeds, but Microsoft Jet records a duplicate key error in the MSysErrors table for each of the records that couldn't be inserted or updated. To correct a duplicate key error, change the value of one of the keys or delete the duplicate record.

Referential Integrity

Referential integrity preserves the relationship between tables when you're adding or deleting records. Enforced referential integrity prevents you from adding a record to or deleting a record from a related table if there is no corresponding record in the primary table. In some situations, enforced referential integrity might result in synchronization

errors. For example, if a user at the Design Master deletes a record with a primary key from a table, and a user at a replica inserts a record that references that key, the next synchronization of the two replicas generates two errors. During synchronization, Microsoft Jet attempts to delete the primary key record at the replica but it can't. The reference to a record or records in a related table at the replica prevents the deletion from succeeding. Microsoft Jet also attempts to add the referencing record at the Design Master but it can't. The new record references a primary key that no longer exists at the Design Master. To correct the two errors, you must delete from the replica the record that references the deleted primary key.

The outcome of the preceding scenario changes if you enforce cascading deletes in your application. With cascading deletes in effect, Microsoft Jet deletes both the primary key record and all referencing records at the Design Master. When you synchronize the Design Master with the replica, the referencing record fails to synchronize because there is no longer a valid primary key; Microsoft Jet records the error in the MSysErrors table. In this situation, however, the error will correct itself during the next synchronization when the replica is notified that the primary key has been deleted from the record.

Using Replication IDs as Primary Keys

To reduce referential integrity errors during synchronization, you may want to use the Replication ID field as the primary key in some or all of your replicated tables. As discussed earlier in this chapter, the Replication ID is a 16-byte GUID that is unique within a table as well as across multiple tables in multiple replicas.

Using the Replication ID field as the primary key is usually better than using an AutoNumber field or custom primary key because it's designed for uniqueness across many replicas. Although a random AutoNumber field may be unique within a table, there is a chance that it may not be unique across multiple replicas.

Assigning the Replication ID field as the primary key in new TableDef objects is easy because there are no related records to update in other tables. Assigning the Replication ID as the primary key in existing tables requires some work because you need to update all foreign keys in related tables to use the Replication ID instead of the existing key. Using DAO, you can write a function that walks through the records of the primary table and the foreign table to reset the foreign key to the Replication ID in the primary table.

Using Replication IDs as primary keys can eliminate "duplicate index" errors but it won't solve referential integrity errors in one-to-many relationships. For example, if a record is deleted from the Products table in one replica and then synchronized with a replica that contains an order referencing that product, you'll get a referential integrity error.

Using the Replication ID field as the primary key might slightly increase replication performance because there is no need to exchange a separate primary key field in addition to a Replication ID. It can also reduce the size of your tables because there is no need to store both a user-defined primary key and a Replication ID. However, it may result in slightly slower query performance because joining tables on 16-byte Replication IDs may take longer than joining tables on 4-byte AutoNumber fields.

Record Locks

If a record is locked when Microsoft Jet attempts to update it during a synchronization, Microsoft Jet retries the update several times. If the record remains locked after repeated attempts, the synchronization fails, and Microsoft Jet records an error in the MSysErrors system table. Although this type of error is exceedingly rare, it might occur in certain multiuser applications. You can ignore errors caused by locked records because Microsoft Jet will retry updating the records during the next synchronization. Because it's unlikely that the same record will be locked during the next synchronization, Microsoft Jet updates the record and removes the error from the MSysError table.

Correcting Synchronization Errors

Synchronization errors are recorded in the MSysErrors table and replicated to all replicas in the replica set. This table includes information about the:

- Table involved.
- Record that encountered the errors.
- Replica or replicas where the error was detected.
- Replica that changed the record last.
- Type of operation that failed.
- Reason it failed.

You must correct errors as soon as possible, because they indicate that the data in different replicas might be diverging. You should be especially careful to correct synchronization errors before moving or renaming your database, because the error is recorded against the Replication ID at the time the error occurred. If the Replication ID changes, Microsoft Jet can't automatically remove the error records during a subsequent synchronization.

In many circumstances, errors are self-correcting during the next synchronization. For example, if you attempt to synchronize a record that another user locked, the update fails. Microsoft Jet records an error and attempts to reapply the update at a later time. If the subsequent update succeeds, the error record is removed. As a general rule,

always synchronize all replicas in the replica set before manually correcting synchronization errors. Due to the nature of bidirectional synchronizations, it might take more than one synchronization to clear the error record from the MSysErrors table after the error is corrected. However, Microsoft Jet should clear all corrected errors from the MSysErrors table after two bidirectional synchronizations.

Design Errors

The order in which you make design or data changes to the database has important implications for synchronizing replicas in the replica set. When you make changes in the design of your database, Microsoft Jet records each change in the MSysSchChange system table. Each change is a separate record in the table, and each record contains all the information about the design change. When Microsoft Jet synchronizes two replicas in the replica set, it compares the MSysSchChange table at the initiating replica with the MSysSchChange table at the target replica. If the two replicas aren't at the same design level, Microsoft Jet applies all the design changes from the most current (highest number of levels) replica to the design of the less current replica. For example, if you change the Design Master six times, each replica must receive all six changes to make its design identical to that of the Design Master. Because of the latency in the synchronization process, one replica might have received only four of the design changes, while another replica might have received all six changes. When the replica having only four changes synchronizes with a replica that has all six changes (or with the Design Master), Microsoft Jet applies the remaining two changes to the design of the less current replica.

When Microsoft Jet applies all the design changes from one replica to another, it applies the changes in the exact order that the changes occurred in the Design Master, and consequently, the order in which the changes were recorded as records in the MSysSchChange table. Microsoft Jet doesn't examine later records in the table to determine whether a design change was undone or further modified by a later change. For example, if you add a table to your database and then make the table replicable, Microsoft Jet records the creation of the table as one design change and makes the table replicable as a second design change. If you then delete the replicable table, Microsoft Jet records that as a third design change. At the next synchronization, Microsoft Jet applies each change in order, even though the table doesn't exist when the synchronization is complete.

Applying design changes to replicas in the exact order the changes occurred at the Design Master ensures that all replicas become identical to the Design Master. However, changes that you or a user make at a replica can cause design errors. For example, if a user at a replica creates a local table and gives it the default name Table1, and you create a replicable table at the Design Master and also give it the default name Table1, Microsoft Jet will fail when it attempts to synchronize the

replica with the Design Master because the replica already has a table with that name. The design error is recorded in the MSysSchemaProb system table and is available for your review.

The MSysSchemaProb table is a local table and is present only when an error occurs when updating the structure of a replica. The table provides details about the cause of the error, including:

- The type of command that failed (for example, a command to create an index or a table).
- The text of the error message.
- The design version that encountered the problem.
- Context information, such as table names and field names.

The rows in the MSysSchemaProb table are automatically deleted when the corresponding design change is successfully applied during synchronization.

To solve design errors, you should carefully review the MSysSchemaProb table to identify the action that failed and then manually remove the element at the replica that is blocking the change. You must always remove the blocking element at the replica even if the design change at the Design Master was the instigator of the error. For example, to correct the error of the two tables named Table1, you must delete the table from the replica even though the table existed before the creation of the table at the Design Master. Removing the blocking element at the Design Master has no effect on correcting the error because the information about the design change is already stored as a record in the MSysSchChange table. Deleting Table1 at the Design Master simply adds a new record to the MSysSchChange table (recording that Table1 was deleted) instead of removing the record that created Table1 in the first place. Because you can't change the contents of the MSysSchChange table, you cannot correct the error except by removing the blocking element at the replica.

Advanced Design Considerations

There are some special factors you should consider when implementing replication. The topics which follow in this section discuss these factors, and provide details on how to address these issues.

Controlling Replica Creation

If you want to control the creation of replicas, you can provide a custom user interface for that purpose. With a custom user interface, your application can track new replicas when they are created. If a user creates a replica outside the control of your application, you can't use DAO code to synchronize the replica because your application won't know about the new replica.

To prevent users from creating a replica of a database by using Microsoft Access menu commands, you can grant them Open privileges for the database when they start your application, and then you can revoke that privilege when they exit your application. This prevents users from opening the database with Microsoft Access when they aren't running your application. Although this isn't an infallible system, it discourages the majority of users from creating replicas.

The following code example grants or revokes the current user's privileges to open the specified database. The application can call this function when a user starts and closes it.

```
Function SetDBAccess(dbs As Database, blnGrantAccess _
   As Boolean) As Integer

   Dim con As Container, doc As Document

   On Error GoTo ErrorHandler

   Set con = dbs.Containers("Databases")

   ' Documents(0) is the document representing
   ' the entire database
   Set doc = con.Documents(0)

   ' Get the currently logged on user. This is
   ' supported only in Microsoft Access.
   doc.UserName = CurrentUser()

   If blnGrantAccess = True Then
      ' Grant Open privileges.
      doc.Permissions = dbSecFullAccess
   Else
      ' Revoke Open privileges.
      doc.Permissions = dbSecNoAccess
   End If

   dbs.Containers.Refresh
   Exit Function

ErrorHandler:
   ' Return error to the caller.
   SetDBAccess = Err
   Exit Function

End Function
```

Using Replicas Instead of Backups

With database replication, it's no longer necessary to make a separate backup copy of your database. If the Design Master is destroyed, you can recover your data from any one of the replicas in the replica set. However, depending on how frequently you synchronize, a replica may not contain all the data in the Design Master or in other replicas. If you want to be able to recover most of the information in your Design Master, you'll need to synchronize frequently.

Although it's possible to back up replicas using traditional backup methods, it's strongly advised not to back up and restore replicas as you would ordinary files. If you back up and restore the Design Master, you could lose critical information about changes to the design of the database as well as the ability of the Design Master to synchronize with the other replicas in the set. If the Design Master is damaged or unusable, don't copy or restore an older version of the Design Master; instead, make another replica the Design Master.

Important A copy of the Design Master created with a backup and restore program may not be able to synchronize with the rest of the replica set.

Designating a New Design Master

The Design Master is the most important replica of a replica set because it's the only replica in which you can make changes to the structure of the database. Under certain circumstances, you might have to move the Design Master to a replica. For example, you might have the Design Master on your computer while another member of your development team has a replica on his or her computer. While you are on vacation, you want the other programmer to be able to make changes to the database. You can use the DesignMasterID property to accomplish the transfer and synchronize the old and new Design Masters, as shown in the following code:

```
Sub SetNewDesignMaster(stroldDM, strNewDM)

   Dim dbs As Database
   Dim dbsNewdmdb As Database

   ' Open the current Design Master in exclusive mode.
   Set dbs = OpenDatabase(stroldDM, True)

   ' Open the database that will become the new
   ' Design Master.
   Set dbsNewdmdb = OpenDatabase(strNewDM)
```

```
    ' Make the new database the Design Master.
    dbs.DesignMasterID = dbsNewdmdb.ReplicaID
    ' Synchronize the old Design Master with the new
    ' Design Master, and allow two-way exchanges.
    dbs.Synchronize strNewDM, dbRepImpExpChanges
    dbs.Close
    dbsNewdmdb.Close

End Sub
```

You must have the Design Master open to make another replica of the new Design Master. The Design Master is, by definition, read/write. If you make a read-only replica the Design Master, it becomes read/write; the old Design Master also remains read/write.

If the Design Master is erased or damaged, you can designate a replica to become the new Design Master. However, remember that you can have only one Design Master at a time.

If you decide to make your own replica the new Design Master for the set, synchronize it with all the replicas in the replica set before setting the DesignMasterID property in your replica. You must have your replica open in exclusive mode to make it the Design Master.

"Cleaning" Records Before Synchronizing

In some applications, you may discard a certain percentage of records because of bad or missing information. For example, if data entry personnel enter information from hard copy forms into a database, a certain percentage of these records must be discarded because they are incomplete. For example, a field in a table may be designed to hold a string of comma-separated numbers, but may instead contain non-comma separators, letters, or other mistakes.

When data is synchronized, all updated data is dispersed, regardless of whether it's useful to a particular application. Bad data in one database becomes bad data in multiple databases when synchronization occurs. Therefore, when building a replication system for your application, you may want to consider writing procedures that either purge or fix data before it's replicated to other systems.

Compacting Your Database Before Synchronizing

Each time you change the design of your Design Master, Microsoft Jet stores information about the change in one of its system tables. If you change the same object more than once before synchronizing, Microsoft Jet continues to store the newest design in preparation for the synchronization. Microsoft Jet replication

therefore makes extensive use of temporary space, especially when you make many design changes to an application. Compacting your database reclaims this temporary space and also reclaims unused space from deleted records. It also arranges database records in sequence, which can improve performance.

For regular databases, compacting once reclaims all unused space. For replicated databases, it's necessary to compact twice in succession in order to reclaim all unused space. Compacting more than twice won't provide any further gains.

Closing Tables Before Synchronizing

Microsoft Jet must open tables exclusively before it can apply design changes to them. If you are using a table in a database at the time you want to synchronize, Microsoft Jet suggests that you first close the table. If you use a "Synchronize Now" button on a form so users can easily click the button whenever they need to synchronize, be sure the button doesn't have an associated table open—either directly or through a query —in the database that is being synchronized.

Synchronizing Before Making Design Changes

When you synchronize two replicas, Microsoft Jet applies the design changes to each replica before applying the data changes. You can decrease the amount of processing during synchronization and reduce the potential for conflicts if you synchronize all the replicas in the set before making design changes in the database.

Binding List Boxes to Replicated Tables

If you bind list boxes to replicated tables, under certain circumstances the list box might display system fields in addition to your regular data fields. If the option to display system objects is turned on, then the system fields in the replication system tables are also displayed. After they're visible, they might appear in the related list box.

CHAPTER 7

Accessing External Data

You can use the Microsoft Jet database engine to access other popular desktop database files, spreadsheets, and textual data stored in tabular format. One way to access external data is through installable ISAM (Indexed Sequential Access Method) drivers.

Supported desktop database installable ISAM (IISAM) formats include older versions of Microsoft Jet, Microsoft FoxPro, dBASE, and Paradox. Supported spreadsheets include Microsoft Excel and Lotus 1-2-3. Microsoft Jet provides additional support for text files in character-delimited and fixed-length formats, as well as Hypertext Markup Language (HTML) files.

This chapter discusses general strategies and techniques for accessing external data through IISAMs and includes specific examples.

Supported External Data Sources

Your Microsoft Jet application can read from and write to (in most cases) external tables, spreadsheets, and text files. You can view and edit external tables with your application without affecting the accessibility from their native applications. If you choose to link an external table, spreadsheet, or text file to a Microsoft Jet database, you can use it just as you would any other table in the database, subject to the limitations imposed by the IISAM driver. You can build queries on the data and create forms and reports that use the data. If the external data source permits multiuser read/write access, you can even view and update the data while other users access the external data source through its native application.

You can use data from any of the following external data sources through the Microsoft Jet database engine.

- Microsoft Jet databases, including databases created with previous versions of Microsoft Jet.
- Microsoft FoxPro databases, versions 2.0, 2.5, 2.6, 3.0, and DBC (database containers).

- dBASE databases, versions III®, dBASE IV®, and dBASE 5.0.
- Paradox databases, versions 3.*x*, 4.*x*, and 5.*x*.
- Microsoft Excel worksheets, versions 3.0, 4.0, 5.0, 7.0, and 8.0.
- Lotus spreadsheets, versions WKS, WK1, WK3, and WK4.
- ASCII text files in tabular format.
- Tabular data in Hypertext Markup Language (HTML) files.
- Folders in Microsoft Exchange mailboxes.

Note Before you can access IISAM data sources, make sure that you have installed the appropriate IISAM drivers. Click the Custom button in Microsoft Visual Basic Setup and select the ODBC and Drivers check box and the Installable ISAMs check box in the Options list box.

External Data Access Requirements

Before you can connect your application to a remote data source, you must make sure that the remote data is accessible to your application's users and that your application can handle remote data source security challenges. You must also make sure that your application interacts correctly with case-sensitive data sources and that the IISAM is correctly initialized for the data source you want to access. Finally, your code shouldn't use objects or methods that are specific to Microsoft Jet data sources when you're accessing non-Microsoft Jet data sources.

Accessing Data Over a Secured Network

As with any other resource on a network, your application must have access to the server and the shared directory where the external data resides. You must determine if your application needs read-only or read/write permissions on the shared directory, and you should establish permissions according to your application's needs.

In addition, you must make sure that your application has proper access to the data through the host application's security system. You can provide password information to the host application directly through Data Access Objects (DAO) when you make your connection.

Overview of IISAM Windows Registry Settings

Each IISAM driver has a corresponding Windows Registry setting that configures the driver upon initialization. During the installation process for the IISAM drivers, the Setup program creates keys in the Windows Registry for the following data sources:

- Microsoft FoxPro 2.0, 2.5, 2.6, 3.0, and DBC
- dBASE III, dBASE IV, and dBASE 5.0
- Paradox, versions 3.*x*, 4.*x*, and 5.*x*

- Microsoft Excel 3.0, 4.0, 5.0, and 8.0
- Lotus WK1, WK3, and WK4
- ASCII text files in tabular format
- HTML files
- Microsoft Exchange 4.0

During this process, the Setup program also assigns a set of default values to each IISAM. Your application can alter these values to change the behavior of certain aspects of each driver. For example, you can select a collating sequence for the FoxPro IISAM that changes the case-sensitive behavior of the driver, as discussed in the following section.

Note The Setup program doesn't write many of the settings for the Microsoft Jet 2.*x* and Microsoft Jet 3.5 IISAM drivers, or for the ODBC drivers. If you make any changes to the Windows Registry settings for any IISAM or for ODBC, you must shut down and restart the application that uses Microsoft Jet so that the new settings can take effect.

Accessing Data on Case-Sensitive Data Sources

You must design your application to support data access operations on external, case-sensitive data sources. Some desktop database products and database servers are case-sensitive by default. These include Microsoft FoxPro, dBASE, and Paradox. In addition, you can configure some database servers that are accessible through ODBC as case-sensitive servers.

If you need to search on a case-sensitive data source, you can design your queries by using Or clauses to capture all possible combinations (all uppercase, all lowercase, or proper-case). However, products that are typically case-sensitive are never case-sensitive in three instances:

- If you're connecting to an Xbase database, such as Microsoft FoxPro or dBASE, check the **CollatingSequence** value in the **\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Jet\3.5\Engines\Xbase** key of the Windows Registry. If this value is set to ascii, all operations are case-sensitive; if it's set to International, all operations aren't case-sensitive.
- If you're connecting to a Paradox database, check the **CollatingSequence** value in the **\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Jet\3.5\Engines\Paradox** key of the Windows Registry. If this value is set to ascii, all operations are case-sensitive; if it's set to International, Norwegian-Danish, or Swedish-Finnish, all operations aren't case-sensitive. The collating sequence for Paradox must match the collating sequence of the .db file that you want to open. If you have a .db file that was created using the ascii collating sequence, setting the **CollatingSequence** value in the Windows Registry to International won't result in case-insensitive sorting.

- If you're connecting to a database server through ODBC, the server's configuration will determine if it's case-sensitive. Check with your database server administrator to determine whether or not your server is case-sensitive.

Note If you run a case-sensitive query across multiple data sources, the collating sequence in effect on the database that hosts the query will determine if the query is case-sensitive.

Unsupported Objects and Methods

Microsoft Jet database applications can't use Microsoft Jet–specific objects and methods with external data sources. The IISAM drivers don't support the following objects and methods:

CompactDatabase method

Container object

CreateDatabase method

CreateField method (You can use this method on new tables, but you can't use it to expand tables with existing fields.)

CreateQueryDef method

Document object

QueryDef object

Relation object

RepairDatabase method

Transaction Support

Although some methods and objects aren't supported through IISAM drivers, the IISAM drivers *do* support transactions on external databases that support transactions. The following table lists the transaction methods that are available on supported databases.

DAO transaction method	Description
BeginTrans	Begins a transaction on the selected data source.
CommitTrans	Completes a transaction on the selected data source by saving changes.
Rollback	Completes a transaction on the selected data source by losing changes.

Establishing Access to External Data

You can access external data tables in three ways: by linking the table in a Microsoft Jet database; by opening the table directly; or by importing the data in the table into a Microsoft Jet database table. The answers to the following questions will help determine the data access method you choose:

- Do you need to maintain access to the data from another application?

  If you need to maintain access to the data from another application, such as the native application used to create the database, you should link or open the external table rather than import the data. Some of the IISAM drivers provide import or export capabilities, but you shouldn't use the import feature unless you are migrating data from another database system, spreadsheet, or textual data source to a Microsoft Jet database.

- Does the data reside on an ODBC data source?

  If the table you want to access resides on an ODBC data source, for best performance you should always link the table rather than attempt to open it directly. In general, you won't need to import data from an ODBC data source.

- Does the data reside on an IISAM data source?

  If the table you want to access resides on an IISAM data source, you can directly open or link the table, depending on your application needs. Linking and opening tables yield approximately the same performance.

- Will you need to establish the connection to the data frequently or just once?

  If you need frequent access to the data source, use a link, because linked tables maintain all connection information between sessions. If you only need occasional access to the data source, you can reduce the size of your Microsoft Jet database by opening the external table directly in code rather than storing the connection information in a link. If you need to access the data source only one time to migrate the data to Microsoft Jet format, use the import capabilities of the IISAM (if available).

The rest of this chapter will discuss accessing external data by either linking or opening an external table.

Important You can't use the Seek method with linked tables.

Linking External Tables

A linked table appears and behaves just like any other table in your Microsoft Jet database (although slight performance differences are associated with connecting to and retrieving remote data). All of the information necessary to establish and maintain a connection to the remote data source is stored within the table definition. Once you link a table, the link is maintained between sessions so it's always available until it's deleted.

▶ To create a link to an external data source

1. Use the CreateTableDef method to create a new TableDef object in the database that will store the link. If this isn't the current database, use the OpenDatabase method to open the database that will contain the link.
2. Set the Connect property of the TableDef object. This property includes an argument that identifies the type of database to which you are connecting. It also includes the path of the file containing the data, and provides authentication services, such as the user name and password to the remote data source, if needed.
3. Set the SourceTableName property of the TableDef object to the name of the table containing the data you want to access.
4. Append the TableDef object to the TableDefs collection.
5. Use the OpenRecordset or CreateQueryDef method on the Database object to open a Recordset or create a query; at the same time, specify the linked table name.

The following code establishes a link to a FoxPro table. It sets the connection information with the Connect and SourceTableName properties and assumes you want to connect to the table Q1Sales in the FoxPro 3.0 database Region1 on the share \\Sales\Regional:

```
Public Sub LinkFoxProTable()

   Dim dbs As Database
   Dim tdfSales As TableDef
   Dim rst As Recordset

   Set dbs = OpenDatabase("Sales.mdb")

   ' Create a new TableDef object in the database
   ' where you want to store the link.
   Set tdfSales = dbs.CreateTableDef _
      ("Western Region Sales")

   ' Set the type of database you are connecting to,
   ' as well as the location of the data source.
   tdfSales.Connect = "FoxPro3.0;" _
      & "DATABASE=\\Sales\Regional\Region1"
```

```
    ' Set SourceTableName to the name of the table
    ' you want to access.
    tdfSales.SourceTableName = "Q1Sales"

    ' Append the TableDef to the TableDefs collection.
    dbs.TableDefs.Append tdfSales

    ' Open a Recordset, specifying the linked table.
    Set rst = dbs.OpenRecordset _
       ("Western Region Sales")

End Sub
```

Important Use a semicolon (;) to separate parameters in the Connect property. Don't include any spaces on either side of the semicolon.

For More Information For more information about the syntax and format of connection strings, see "Specifying Connection Information" later in this chapter, or see "Connect Property" or "SourceTableName Property" in the *Language Reference* in Books Online.

You can set the *source* and *connect* values as arguments in the CreateTableDef method instead of setting them separately as properties of the TableDef object. The following example is similar to the previous example, but it sets the connection and table information in the CreateTableDef method:

```
Public Sub LinkFoxProTable2()

   Dim dbs As Database
   Dim tdfSales As TableDef

   Set dbs = OpenDatabase("Sales.mdb")
   ' Set the connection and table information with
   ' CreateTableDef.
   Set tdfSales = dbs.CreateTableDef _
      ("Western Region Sales", 0, "Q1Sales", _
      "FoxPro3.0;" _
      & "DATABASE=\\Sales\Regional\Region1;")

   dbs.TableDefs.Append tdfSales

End Sub
```

Opening External Tables

In contrast to linking, when you open a database directly, you must supply the connection information at the beginning of each session to establish a connection to the data source. None of the information needed to establish a connection to the remote data source is stored in your Microsoft Jet database.

In this case, you set the *source* and *connect* values as arguments in the OpenDatabase method instead of setting them separately as properties of a TableDef object. The following code fragment shows how to open a database directly instead of linking a table:

```
Public Sub OpenFoxProTable()

   Dim dbs As Database
   Dim rstAuthors As Recordset

   ' Open an external Paradox database named Publish.
   Set dbs = OpenDatabase("C:\PDX\Publish", _
      False, False, "Paradox 4.X;")

   ' Create a Recordset from the Authors table.
   Set rstAuthors = dbs.OpenRecordset("Authors")

End Sub
```

Note Use the *dbname* argument of the OpenDatabase method to specify the database name, and the *source* argument of the OpenRecordset method to specify the table name for all IISAM drivers. For external data sources that store one table per file, the *dbname* argument should contain the full directory or network path to the file, and *source* should contain the data file name. For Paradox databases, the data file name should have no extension. You can, however, include an extension for dBASE and FoxPro. For external data sources that store multiple tables in a file, *dbname* should contain the full directory or network path to the file including the file name and extension; *source* should contain the table name.

You can specify a *type* argument for the OpenRecordset method if you want more control over the Recordset type. By default, Microsoft Jet opens the highest-performance Recordset type available for the data source you're accessing. If you open the data source directly, Microsoft Jet will use dbOpenTable.

Note The dbOpenTable constant isn't a valid argument for ODBC databases and linked IISAM tables.

After you've opened the table, you can access the Recordset object you created just as you would any other Recordset in your Microsoft Jet database. Keep in mind, however, that you must re-establish your connection to the external data source and re-create the Recordset object for each session.

For More Information For more information about working with Recordset objects, see Chapter 3, "Working with Records and Fields," or see "Recordset Object" in the *Language Reference* in Books Online.

Specifying Connection Information

Earlier in this chapter, several examples of *connection strings* showed how to specify connection information. One of these examples was:

```
tdfSales.Connect = _
   "FoxPro 3.0;DATABASE=\\Sales\Regional\Region1"
```

The following sections contain additional information about connection strings, their syntax, and their format.

Specifying Database Names

When you specify a database name for a database, spreadsheet, or text file supported by an IISAM, you must include the fully qualified path to the file. On a local drive, you must include the drive letter, directory path, and file name; on a network drive, you must include the server name, share, directory path, and file name. You can also map a server, share, and directory path to a logical drive and indicate the database name as the drive letter followed by the file name. If you are accessing data on the Internet, you can specify the FTP location of the file. The following examples illustrate each of these situations.

To specify the FoxPro database Region1 in the \FoxPro3 directory on the local C drive, use the following database name:

```
C:\FoxPro3\Region1
```

To specify the FoxPro database Region1 on the Regional share on the Sales server, use the following database name:

```
\\Sales\Regional\Region1
```

To specify the FoxPro database Region1 on the Regional share on the Sales server where the network path \\Sales\Regional has been assigned to drive G, use the following database name:

```
G:\Region1
```

To specify the FoxPro database Region1 in the directory Regional on the FTP server named Sales, use the following database name:

```
ftp://Sales/Regional/Region1
```

Note ODBC data sources use a different specifier, called a data source name (DSN), to indicate the database name. For all ODBC data sources, use an empty string ("") as the database name. The ODBC DSN is described in the following section, "Specifying Data Source Names."

If you specify an FTP location, the target database on the FTP server is automatically copied into a new subdirectory of the temporary directory specified in the TEMP environment variable. The new subdirectory is named Net0000.tmp.

Specifying Data Source Names

ODBC data sources use a different syntax for specifying database names and connection information. You can add the data source name, or DSN, by using the ODBC Administrator application or the RegisterDatabase method of the DBEngine object. Data source names are stored in the Windows Registry key **\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\ODBC\ODBC.INI.**

Each entry in the **ODBC.INI** key assigns a logical DSN to a set of attributes that includes: the name of the ODBC database server; the name of the database on the server, if multiple databases are supported; the type of server (for example, SQL Server); a description of network and connection parameters; and additional information, such as character set conversions. Because connections to ODBC data sources rely on this information, your application must make sure users have the appropriate DSN in their Windows Registry before you attempt to access remote data.

Specifying Source Database Types

Each external ISAM format has an associated database type that you must specify when using the driver. Each IISAM driver also has a Windows Registry key that configures the behavior of the driver. To build a connection string, you must include a source database type so that Microsoft Jet knows how to handle the external data.

The following table lists the source database types.

Data source	Source database types
dBASE	dBASE III dBASE IV dBASE 5.0
Microsoft Excel	Excel 3.0 Excel 4.0 Excel 5.0 (used for Excel 5.0 and Excel 7.0) Excel 8.0
FoxPro	FoxPro 2.0 FoxPro 2.5 FoxPro 2.6 FoxPro 3.0 FoxPro DBC
Lotus	Lotus WK1 Lotus WK3 Lotus WK4
ODBC	ODBC
Paradox	Paradox 3.*x* Paradox 4.*x* Paradox 5.*x*

Data source	Source database types
Text	Text
HTML	HTML Import
Microsoft Exchange	Exchange 4.0

Note You must enter the source database type strings exactly as they appear in the table, including spaces and punctuation.

Specifying Passwords

Microsoft Jet honors the security system in all external data sources that support security. Your application must supply a password to the external data source to establish a connection. You can do this by prompting the user for this information and building it into the connection string, or you can include the password in your application's code.

Note If you store sensitive data in the external data source, you might not want to use hard-coded passwords in your Microsoft Jet application. If you do so, consider encrypting your application so that others can't discover the password with a text editor or disk utility.

Use the PWD parameter to send the password to the external data source, as in the following example. You must follow the password parameter with a semicolon.

```
Set tdfRegionOne = dbs.CreateTableDef _
   ("First Quarter Sales",0,"Q1Sales", _
   "Paradox 4.X;DATABASE=" _
   & "\\Sales\Regional\Region1;PWD=RollsRoyce;")
```

You can use the password parameter to specify database passwords only; you can't use the password to log on to a network. If your application requires network access, you must establish these connections before attempting to access the external data.

You can't use the password parameter to decrypt Microsoft Excel worksheets or workbooks; you must unprotect and save the worksheet or workbook in Microsoft Excel before you can open it with the Microsoft Excel IISAM driver.

Saving ODBC Passwords Between Sessions

If you don't want to use hard-coded passwords for ODBC databases in your application or require your users to enter a password each time they start the application, you can set the Attributes property of the TableDef object to dbAttachSavePWD. This saves the ODBC password as part of the table definition. Each time a user opens the linked table, the password information is automatically applied. The following line of code shows how to save a password:

```
tdfRegionOne.Attributes = dbAttachSavePWD
```

If your remote data is particularly sensitive, you may not want to allow users to save their passwords. To change this behavior, you must modify the MSysConf table in your remote database to forbid local storage of logon IDs and passwords. ODBC will automatically prompt users for any missing authentication information when they attempt to open the remote database.

For More Information For more information about the MSysConf table, see Chapter 9, "Developing Client/Server Applications."

Working with Desktop Databases

Earlier in this chapter, several examples showed how to access desktop databases. The following sections provide more detailed information about accessing data in Borland dBase, Microsoft FoxPro, Borland Paradox, and previous versions of Microsoft Jet databases.

Working with Microsoft Jet Databases

You can access data in other Microsoft Jet databases (.mdb), including databases created with previous versions of Microsoft Jet.

Linking External Microsoft Jet Tables

To link an external Microsoft Jet table in your local Microsoft Jet database, use the OpenDatabase method to open the local Microsoft Jet database, create a TableDef object, and set the Connect and SourceTableName properties of the TableDef object to indicate the path and name, respectively, of the external Microsoft Jet table you want to link. This method works for Microsoft Jet 1.*x*, 2.*x*, and 3.*x* databases.

The following example links an external Microsoft Jet table and displays a message box as confirmation. Notice that the connection string begins with just a semicolon (;) rather than a source database type. A source database type isn't applicable when connecting to external databases that use the Microsoft Jet database engine.

```
Public Sub LinkJetTable()

   Dim dbsJet As Database
   Dim tdfExternalJetTable As TableDef

   ' Open a Microsoft Jet database.
   Set dbsJet = OpenDatabase _
      ("C:\Jet_Samp.mdb")

   ' Create a TableDef object.
   Set tdfExternalJetTable = dbsJet.CreateTableDef _
      ("Linked Jet Table")
```

```
' Set the connection information.
   tdfExternalJetTable.Connect = ";DATABASE=" _
      & "\\Access\Data\AP.mdb"
   tdfExternalJetTable.SourceTableName = "Accounts"

   ' Append the TableDef object to create a link.
   dbsJet.TableDefs.Append tdfExternalJetTable

   ' Display the confirmation message.
   MsgBox "Finished linking " & _
   tdfExternalJetTable.SourceTableName & ".", 0

End Sub
```

Opening External Microsoft Jet Tables

To open an external Microsoft Jet table directly, use the OpenDatabase method to indicate the path, exclusive access value, read-only value, and data source name.

The following example opens an external Microsoft Jet table and then creates a Recordset object from the table:

```
Public Sub OpenJetTable()

   Dim dbsExternalJet As Database
   Dim rstAccounts As Recordset

   ' Open an external Microsoft Jet
   ' database named AP.
   Set dbsExternalJet = OpenDatabase _
      ("\\Access\Data\AP.mdb", False, False, "")

   ' Open a Recordset on the Accounts table.
   Set rstAccounts = dbsExternalJet. _
      OpenRecordset("Accounts")

End Sub
```

Working with dBASE Databases

The Microsoft Jet Xbase IISAM driver supports the following versions of Borland's dBASE product line: dBASE III, dBASE IV, and dBASE 5.0. When you're accessing dBASE databases through the Microsoft Jet database engine, you should also consider how Microsoft Jet handles Paradox data types, deleted records, dBASE indexes, and Memo fields.

When specifying connection information for dBASE data, use the following specifications.

For	Use
Source database type	One of the following: • dBASE III • dBASE IV • dBASE 5.0
Database name	The full path to the directory containing the dBASE files you intend to access.
Source table name	The name of the table.

Linking dBASE Tables

To link a dBASE database to your Microsoft Jet database, use the OpenDatabase method to open the local Microsoft Jet database, create a TableDef object, and set the Connect and SourceTableName properties of the TableDef object to indicate the path and name, respectively, of the dBASE table that you want to link.

The following example links an external dBASE IV table and then displays a message box as confirmation:

```
Public Sub LinkDBaseTable()

   Dim dbsJet As Database
   Dim tdfDBaseTable As TableDef

   ' Open a Microsoft Jet database.
   Set dbsJet = OpenDatabase _
      ("C:\Jet_Samp.mdb")

   ' Create a TableDef object.
   Set tdfDBaseTable = dbsJet.CreateTableDef _
      ("Linked dBASE Table")

   ' Set the connection information.
   tdfDBaseTable.Connect = _
      "dBASE IV;DATABASE=\\dBASE\Data\AP"
   tdfDBaseTable.SourceTableName = "Accounts"

   ' Append the TableDef object to create the link.
   dbsJet.TableDefs.Append tdfDBaseTable

   ' Display the confirmation message.
   MsgBox "Finished linking " & _
      tdfDBaseTable.SourceTableName & ".", 0

End Sub
```

Opening dBASE Tables

To open a dBASE table directly, use the OpenDatabase method to indicate the path, exclusive access value, read-only value, and data source name.

The following example opens an external dBASE III table directly and creates a Recordset object from the table:

```
Public Sub OpenDBaseTable()

   Dim dbsdBASE As Database
   Dim rstAccounts As Recordset

   ' Open the external dBASE database.
   Set dbsdBASE = OpenDatabase _
      ("\\dBASE\Data\AP", False, False, _
      "dBASE III;")

   ' Create a Recordset object from
   ' the Accounts table.
   Set rstAccounts = dbsdBASE. _
      OpenRecordset("Accounts")

End Sub
```

Converting dBASE Data Types to Microsoft Jet Data Types

Microsoft Jet translates each dBASE data type into a corresponding Microsoft Jet data type when your application reads the data. The following table shows the one-to-one correspondence between data types.

DBASE data type	Microsoft Jet data type
Character	Text
Numeric, Float	Double
Logical	Boolean
Date	Date/Time
Memo	Memo
OLE	Long Binary

Handling Deleted Records

When you delete a record from a dBASE table through your application, the record is marked as deleted in the table. The record isn't, however, removed from the table because dBASE doesn't recover deleted records until the table is compacted. Because you can't compact a dBASE table through Microsoft Jet, you must compact the dBASE database to physically remove the deleted records.

Although records marked for deletion aren't actually removed in the source database, they won't appear in your application because Microsoft Jet auto-

matically filters out deleted records. If you want records marked for deletion to appear in your data set, you must set the Windows Registry **Deleted** value in the **\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Jet\3.5\Engines\Xbase** key to False. Then, you must restart the application that uses Microsoft Jet to reflect the record deletions.

Handling dBASE Indexes

To speed up Microsoft Jet access to dBASE tables, specify the dBASE .ndx or .mdx index files to use. You can specify index files in an .inf file by creating a text file with the same file name as your dBASE database and appending an .inf extension. For Microsoft Jet to easily locate associated indexes, it's best to put the .inf file in the same directory as the associated table. However, if the table is on a read-only share, you can specify a different path using the **INFPath** value in the **\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Jet\3.5\Engines\Xbase** key in the Windows Registry.

The following procedure describes how to create an .inf file for the hypothetical dBASE database called Sales.dbf.

1. Create the text file Sales.inf.
2. On the first line, add the database type identifier enclosed in square brackets. For example, to specify dBASE III indexes, type the following:

   [dBASE III]

3. Assign an index number to the first index for the database. Number the first index NDX1, the second NDX2, and so on. If you want to specify multiple index files (.mdx) for dBASE IV or dBASE 5.0 databases, use index numbers of the form MDX1, MDX2, and so on. You can also specify unique indexes by prepending the string "UIDX" to the index number. For MDX indexes, prepend the string "MDX" to the index number.
4. Follow the index number with an equal sign and the file name of the index that you want to specify, including the extension. Don't put any spaces in this entry. For example, to specify the index custname.ndx as the first index and the unique index custno.ndx as the second index, add the following entries:

   NDX1=custname.ndx

   UIDX1=custno.ndx

5. Repeat steps 3 and 4 until you've specified all indexes for the database table. Your completed .inf file should look similar to the following:

```
[dBASE III]
NDX1=custname.ndx
UIDX1=custno.ndx
```

6. Save the .inf file in the directory that contains your dBASE database. Alternatively, you can create a Windows Registry entry to point to the path where the .inf file is stored. This is useful if you can't store the .inf file in the same directory as the database or if you want to keep all .inf files in one shared location for easy maintenance.

   If you want to create an **INFPath** entry in your Windows Registry, you must manually add the Registry value in the Registry Editor as a string value. To add the value, open the **\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Jet\3.5\Engines\Xbase** key in the Windows Registry; on the Edit menu, point to New, and then click String Value. Type **INFPath** as the name, and type the full directory or network path to the .inf file in the **Data** column. Save the Registry changes and restart the application that uses Microsoft Jet to use this setting.

Keep in mind that Microsoft Jet will need to periodically update your index files as you change data in your dBASE database. If you associate index files with a dBASE database that is linked or opened from a Microsoft Jet database, you need to make sure the indexes are also available to your Microsoft Jet database.

Important Don't delete or move the index files or the .inf file without updating associated references (such as the Windows Registry). If Microsoft Jet doesn't have access to up-to-date index information, it can't process queries correctly.

Handling Memo Fields

Memo fields for dBASE databases are stored in the file system instead of in the database. If you want to link or open a dBASE database that has Memo fields, make sure the Memo files are stored in the same directory as the database files.

Working with FoxPro Databases

The Microsoft Jet Xbase IISAM driver supports the following versions of Microsoft FoxPro: 2.0, 2.5, 2.6, 3.0, and DBC. When you're accessing FoxPro databases through the Microsoft Jet database engine, you should also consider how Microsoft Jet handles FoxPro data types, deleted records, FoxPro indexes, and Memo fields.

Note You can't link or open .dbf files directly in FoxPro version 3.0. To access data in FoxPro version 3.0 databases, you must import the data into a Microsoft Jet database, manipulate the data, and save it as a stand-alone .dbf file (referred to as a "free table" in FoxPro). FoxPro users can then open the .dbf file as a stand-alone database, but they can't use it in conjunction with a .dbc file.

When specifying connection information for FoxPro data, use the following specifications.

For	Use
Source database type	One of the following: • FoxPro 2.0 • FoxPro 2.5 • FoxPro 2.6 • FoxPro 3.0 • FoxPro DBC
Database name	The full path to the directory containing the FoxPro files that you intend to access. (For Microsoft FoxPro DBC, the path must include the name of the .dbc file.)
Source table name	The name of the table. (Use the .dbf file name without the extension, or the complete file name with the extension, but substitute a number sign [#] for the dot [.] that precedes the file name extension; for Microsoft FoxPro DBC, use the table name in the DBC.)

Linking FoxPro Tables

To link a FoxPro database to your Microsoft Jet database, use the OpenDatabase method to open the local Microsoft Jet database, create a TableDef object, and set the Connect and SourceTableName properties of the TableDef object to indicate the path and name, respectively, of the FoxPro table that you want to link.

The following example links an external FoxPro version 2.6 table and then displays a message box as confirmation. Note that this example uses a number sign (#) in the SourceTableName property to specify a nonstandard extension (.xyz).

```
Public Sub LinkFoxProTable()

   Dim dbsJet As Database
   Dim tdfFoxTable As TableDef

   ' Open a Microsoft Jet database.
   Set dbsJet = OpenDatabase _
      ("C:\Jet_Samp.mdb")

   ' Create a TableDef object.
   Set tdfFoxTable = dbsJet.CreateTableDef _
      ("Linked FoxPro Table")

   ' Set the connection information.
   tdfFoxTable.Connect = _
      "FoxPro 2.6;DATABASE=\\FoxPro\Data\AP"
   tdfFoxTable.SourceTableName = "Accounts#xyz"
```

```
    ' Append the TableDef object to create the link.
    dbsJet.TableDefs.Append tdfFoxTable
    ' Display the confirmation message.
    MsgBox "Finished linking " _
       & tdfFoxTable.SourceTableName & ".", 0

End Sub
```

Opening FoxPro Tables

To open a Microsoft FoxPro table directly, use the OpenDatabase method to indicate the path, exclusive access value, read-only value, and data source name.

The following example opens an external FoxPro version 2.0 table and then creates a Recordset object from the table:

```
Public Sub OpenFoxProTable()

    Dim dbsFox As Database
    Dim rstAccounts As Recordset

    ' Open an external FoxPro database.
    Set dbsFox = OpenDatabase _
       ("\\FoxPro\Data\AP", False, False, _
       "FoxPro 2.0;")

    ' Create a Recordset object from
    ' the Accounts table.
    Set rstAccounts = dbsFox.OpenRecordset("Accounts")

End Sub
```

Converting FoxPro Data Types to Microsoft Jet Data Types

Microsoft Jet translates each FoxPro data type into a corresponding Microsoft Jet data type when your application reads the data. The following table shows the one-to-one correspondence between data types.

FoxPro data type	Microsoft Jet data type
Character	Text
Numeric, Float	Double
Logical	Boolean
Date	Date/Time
Memo	Memo
General	Long Binary
Double*	Double

FoxPro data type	Microsoft Jet data type
Currency*	Currency
Integer*	Long
DateTime*	Date/Time

* Supported in Microsoft FoxPro 3.0 and Microsoft FoxPro DBC only.

Handling Deleted Records

When you delete a record in a FoxPro table from your application, the record is marked as deleted in the table. The record isn't, however, removed from the table because FoxPro doesn't recover deleted records until the table is compacted. Because you can't compact a FoxPro table through Microsoft Jet, you must compact the FoxPro database to physically remove the deleted records.

Although records marked for deletion aren't actually removed in the source database, they won't appear in your application because Microsoft Jet automatically filters out deleted records. If you want records marked for deletion to appear in your data set, you must set the Windows Registry **Deleted** value in the **\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Jet\3.5\Engines\Xbase** key to False. Then, you must restart the application that uses Microsoft Jet to reflect the record deletion.

Handling FoxPro Indexes

You can speed up Microsoft Jet access to FoxPro tables by specifying which FoxPro .idx or .cdx index files to use. You specify index files in an .inf file, which you can create by making a text file with the same file name as your FoxPro database and appending an .inf extension. The following procedure describes how to create an .inf file for a FoxPro database called Sales.dbf.

1. Create the text file Sales.inf.
2. On the first line of the text file, add the database type identifier enclosed in square brackets. For example, to specify FoxPro 2.0 indexes, type the following:

   [FoxPro 2.0]

3. Assign an index number to the first index for the database. Number the first index IDX1, the second IDX2, and so on. If you want to specify combined index files (.cdx) for FoxPro databases, use index numbers of the form CDX1, CDX2, and so on. You can also specify unique indexes by prepending a "U" to the index number.

4. Follow the index number with an equal sign and the file name of the index you want to specify, including the extension. Don't put any spaces in this entry. For example, to specify the index custname.idx as the first index and the unique index custno.idx as the second index, add the following entries:

   IDX1=custname.idx

   UIDX1=custno.idx

5. Repeat steps 3 and 4 until you have specified all indexes for the database table. Your completed .inf file should look similar to the following:

```
[FoxPro 2.0]
IDX1=custname.idx
UIDX1=custno.idx
```

6. Save the .inf file in the directory that contains your FoxPro database. Alternatively, you can create a Windows Registry entry to point to the path where the .inf is stored. This is useful if you can't store the .inf file in the same directory as the database or if you want to keep all .inf files in one shared location for easy maintenance.

   If you want to create an **INFPath** entry in your Windows Registry, you must manually add the Registry value in the Registry Editor as a string value. To add the value, open the **\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Jet\3.5\Engines\Xbase** key in the Windows Registry, on the **Edit** menu, point to **New**, and then click **String Value**. Type **INFPath** as the name and type the full directory or network path to the .inf file in the **Data** column. Save the Registry changes and restart the application that uses Microsoft Jet to use this setting.

Keep in mind that Microsoft Jet will need to periodically update your index files as you change data in your FoxPro database. If you associate index files with a FoxPro database that is linked or opened from a Microsoft Jet database, you must make sure that the indexes are also available to your Microsoft Jet database.

Note Don't delete or move the index files or the .inf file without updating associated references (such as INFPath entry in the Windows Registry). If Microsoft Jet doesn't have access to up-to-date index information, it can't process queries correctly.

Handling Memo Fields

Memo fields for FoxPro databases are stored in the file system rather than in the database. If you want to link or open a FoxPro database that has Memo fields, make sure the Memo files are stored in the same directory as the database files.

Working with Paradox Databases

The Microsoft Jet Paradox IISAM driver supports the following versions of Borland's Paradox product line: Paradox 3.*x*, Paradox 4.*x*, and Paradox 5.*x*. When you're accessing Paradox databases through the Microsoft Jet database engine, you should also consider how Microsoft Jet handles Paradox data types and Paradox indexes and keys. In addition, you should understand how Microsoft Jet uses Windows Registry settings to control access to Paradox tables in a network environment.

When specifying connection information for Paradox tables, use the following specifications.

For	Use
Source database type	One of the following: • Paradox 3.*x* • Paradox 4.*x* • Paradox 5.*x*
Database name	The full path to the directory containing the Paradox files that you intend to access.
Source table name	The name of the table (the .db file name without the extension).

Linking Paradox Tables

To link a Paradox database to your Microsoft Jet database, use the OpenDatabase method to open the local Microsoft Jet database, create a TableDef object, and set the Connect and SourceTableName properties of the TableDef object to indicate the path and name, respectively, of the Paradox table that you want to link.

The following example links an external Paradox version 3.*x* table and then displays a message box as confirmation:

```
Public Sub LinkParadoxTable()

   Dim dbsJet As Database
   Dim tdfParadox As TableDef

   ' Open a Microsoft Jet database.
   Set dbsJet = OpenDatabase _
      ("C:\Jet_Samp.mdb")

   ' Create a TableDef object.
   Set tdfParadox = dbsJet.CreateTableDef _
      ("Linked Paradox Table")
```

```
    ' Set the connection information.
    tdfParadox.Connect = "Paradox 3.X;" _
       & "DATABASE=\\PDOX\Data\AP"
    tdfParadox.SourceTableName = "Accounts"
    ' Append a TableDef object to create the link.
    dbsJet.TableDefs.Append tdfParadox

    ' Display a confirmation message.
    MsgBox "Linking done!" _
       & tdfParadox.SourceTableName & ".", 0

End Sub
```

Opening Paradox Tables

To open a Paradox table directly, use the OpenDatabase method to indicate the path, exclusive access value, read-only value, and data source name.

The following example opens an external Paradox version 4.*x* table and then creates a Recordset object from the table.

```
Public Sub OpenParadoxTable()

   Dim dbsParadox As Database
   Dim rstAccounts As Recordset

   ' Open an external Paradox database named AP.
   Set dbsParadox = OpenDatabase _
      ("\\PDOX\Data\AP", False, False, _
      "Paradox 4.X;")

   ' Create a Recordset from the Accounts table.
   Set rstAccounts = dbsParadox.OpenRecordset _
      ("Accounts")

End Sub
```

Converting Paradox Data Types to Microsoft Jet Data Types

Microsoft Jet translates each Paradox data type into a corresponding Microsoft Jet data type when your application reads the data. The following table shows the one-to-one correspondence between data types.

Paradox data type*	Microsoft Jet data type
Alphanumeric	Text
Number	Double
Short number	Integer
Currency	Double

Paradox data type*	Microsoft Jet data type
Date	Date/Time
Memo	Memo
OLE	Long Binary (The Microsoft Jet database engine recognizes the object but won't let you open it.)
Logical (Paradox 5.*x*)	Boolean
Integer (Paradox 5.*x*)	Long
Timestamp (Paradox 5.*x*)	Date/Time
Binary (Paradox 5.*x*)	Byte
BCD (Paradox 5.*x*)	Double
Time (Paradox 5.*x*)	Date/Time
Autoinc (Paradox 5.*x*)	AutoNumber

* Paradox Graphic, Binary, and Formatted Memo types aren't supported and won't appear when you display external Paradox tables.

Handling Paradox Indexes and Keys

Paradox uses a .px index file to store primary key information for each table that has a primary key. Microsoft Jet must be able to locate the .px file each time you attempt to use the table. If the .px file is unavailable, Microsoft Jet will be unable to open the Paradox table.

You can't update data in a Paradox table that has no primary key. Also, you can't open a Paradox table that has no primary key in shared mode.

Understanding the ParadoxNetPath, ParadoxUserName, and ParadoxNetStyle Settings

Microsoft Jet uses three settings — ParadoxNetPath, ParadoxUserName, and ParadoxNetStyle — to control user access to Paradox tables that reside on a file server. You can set each of these values in the Windows Registry.

When you link or open a Paradox table that resides on a server and is shared by multiple users, you must set the **ParadoxNetPath** option in the **\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Jet\3.5\Engines\Paradox** key of the Windows Registry to the path for either the Paradox.net file (for Paradox 3.*x*) or the Pdoxusrs.net file (for Paradox 4.*x*). For example, if the Paradox.net file is on drive Q in the Wrkgrp directory, set **ParadoxNetPath** as follows:

```
ParadoxNetPath=Q:\Wrkgrp
```

The **ParadoxUserName** value is a string value that Paradox displays to an interactive Paradox user when the user attempts to place a lock that is incompatible with the lock held by the Microsoft Jet Paradox IISAM driver.

The **ParadoxNetStyle** value indicates whether users are accessing the Paradox database using the Paradox 3.*x* or Paradox 4.*x* locking method.

Note The ParadoxNetPath value, including the drive letter, must be consistent for all users sharing a particular database (directory). For example, if the Paradox.net file is in the Wrkgrp directory on drive Q for one user of the database, it must be in the Wrkgrp directory on drive Q for all other users of the database.

Working with Spreadsheets

With Microsoft Jet and IISAM drivers, you can open spreadsheets and manipulate their information as if they were database tables. The steps to do this are very similar to the steps for connecting to external databases. The following sections explain how to use Data Access Objects (DAO) to access Microsoft Excel worksheets and workbooks and Lotus 1-2-3 spreadsheets.

Both spreadsheet products have one common feature worth mentioning: the HDR parameter in the connection string. The following sections discuss this and additional considerations for each spreadsheet product.

Using the HDR Parameter to Suppress Headers

The Windows Registry keys that manage behavior in Microsoft Excel worksheets and workbooks, and Lotus 1-2-3 spreadsheets, contain a setting that determines whether the IISAM should use the data in the cells of the first row of the range as column names in the Microsoft Jet database. You can find this setting, FirstRowHasNames, in the **\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Jet\3.5\Engines\Lotus** key for Lotus 1-2-3 and the **\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Jet\3.5\Engines\Excel** key for Microsoft Excel. When the value of **FirstRowHasNames** is 01 (True, the default), *all* ranges of that product family are handled as if the first row contained column name information.

To override this Windows Registry setting on a range-by-range basis, you must use the HDR parameter in the connection string. Set HDR to True (`HDR=Yes`) to instruct the IISAM to use the data in the first row of the sheet as column names, regardless of the Registry setting. Set HDR to False (`HDR=No`) to instruct the IISAM to treat the first row of the sheet as data, regardless of the Registry setting.

Working with Microsoft Excel Worksheets and Workbooks

The Microsoft Jet IISAMs support the following single-sheet worksheet and multiple-sheet workbook versions of Microsoft Excel: Excel 3.0 and Excel 4.0 for single-sheet worksheets, and Excel 5.0 (for Microsoft Excel 5.0 and 7.0) and Excel 8.0 for multiple-sheet workbooks. There are a few operations that you can't perform on Microsoft Excel worksheets or workbooks through the Microsoft Excel IISAM:

- You can't delete rows from Microsoft Excel worksheets or workbooks.
- You can clear data from individual cells in a worksheet, but you can't modify or clear cells that contain formulas.
- You can't create indexes on Microsoft Excel worksheets or workbooks.
- You can't read encrypted data through the Microsoft Excel IISAM. You can't use the PWD parameter in the connection string to open an encrypted worksheet or workbook, even if you supply the correct password. You must decrypt all Microsoft Excel worksheets or workbooks through the Microsoft Excel user interface if you plan to link or open them in your Microsoft Jet database.

Specifying Sheets and Ranges in Microsoft Excel Worksheet and Workbook Files

You can specify a subset of the available data when you open a Microsoft Excel worksheet or workbook. In worksheet files, you can open the entire sheet, a named range of cells, or an unnamed range of cells. In a workbook file, you can open a single worksheet, a named range anywhere in the workbook, or an unnamed range in a single worksheet. The following table lists the conventions for the DATABASE and *source* arguments you must supply in the connection string to access each of these objects.

To access this object	In this version of Microsoft Excel	Use this syntax
Entire sheet in a worksheet file	3.0 and 4.0	Use the DATABASE parameter to specify the fully qualified network or directory path to the worksheet file; use the *source* argument to specify the sheet as *filename*#**xls**, where *filename* is the name of the worksheet.

To access this object	In this version of Microsoft Excel	Use this syntax
Entire worksheet in a workbook file	5.0, 7.0, and 8.0	Use the DATABASE parameter to specify the fully qualified network or directory path to the workbook file, including the workbook file name; use the *source* argument to specify the sheet as *sheetname*$, where *sheetname* is the name of the worksheet. **Important** You must follow the worksheet name with a dollar sign ($).
Named range of cells in a worksheet or workbook file	3.0, 4.0, 5.0, 7.0, and 8.0	Use the DATABASE parameter to specify the fully qualified network or directory path to the worksheet or workbook file, including the worksheet or workbook file name; use the *source* argument to specify the named range as *NamedRange*, where *NamedRange* is the name you assigned to the range in Microsoft Excel. **Important** You must name the range in Microsoft Excel before attempting to open or link it.
Unnamed range of cells in a worksheet file	3.0 and 4.0	Use the DATABASE parameter to specify the fully qualified network or directory path to the worksheet file, including the worksheet file name; use the *source* argument to specify the range as **A1:Z256**. Replace **A1:Z256** with the range of cells you want to access.
Unnamed range of cells in a single worksheet in a workbook file	5.0 and 7.0	Use the DATABASE parameter to specify the fully qualified network or directory path to the workbook file, including the workbook file name; use the *source* argument to specify the sheet you want to link or open as *sheetname*$ and the range as **A1:Z256**. For example, to access cells A1 through Z256 in worksheet SheetName, you would use the following in the *source* argument `SheetName$A1:Z256`.

Note You can't specify a value in a range that exceeds the maximum number of rows, columns, or sheets for the worksheet or workbook. See your Microsoft Excel documentation for these values.

Linking Microsoft Excel Spreadsheets

To link a Microsoft Excel spreadsheet to your Microsoft Jet database, use the OpenDatabase method to open the local Microsoft Jet database, create a TableDef object, and set the Connect and SourceTableName properties of the TableDef object to indicate the Microsoft Excel spreadsheet that you want to link.

The following example links an external Microsoft Excel spreadsheet:

```
Public Sub LinkExcelSheet()

   Dim dbsJet As Database
   Dim rstSales As Recordset
   Dim tdfExcelSheet As TableDef

   Set dbsJet = OpenDatabase("C:\Jet_Samp.mdb")

   ' Create a TableDef object.
   Set tdfExcelSheet = dbsJet.CreateTableDef _
      ("Linked Excel Worksheet")

   ' Set connection information.
   tdfExcelSheet.Connect = "Excel 5.0;" _
      & "Database=C:\Sales\Q1Sales.xls"
   tdfExcelSheet.SourceTableName = "January Sales"

   ' Append the TableDef object to create a link.
   dbsJet.TableDefs.Append tdfExcelSheet

   ' Open a Recordset object on the Microsoft Excel
   ' worksheet
   Set rstSales = dbsJet.OpenRecordset _
      ("Linked Excel Worksheet")

End Sub
```

Opening Microsoft Excel Spreadsheets

The following code examples illustrate how to specify an entire worksheet, a named range, and an unnamed range of cells in a Microsoft Excel 4.0 worksheet and a Microsoft Excel 7.0 workbook.

This first example opens a Microsoft Excel 4.0 worksheet from a Microsoft Jet database three times to demonstrate how to open an entire worksheet, a named range in a worksheet, and an unnamed range in a worksheet. The example tests the ranges by counting the number of records that appear in the Recordset.

Because the HDR parameter in the connection string is set to No, the first row of the spreadsheet is treated as a record in the table, not as a header containing column names. If the HDR parameter was set to Yes in this example, the total records in the Recordset wouldn't include the first record, and the variable `intNumRecords` would be one less than its value when HDR is set to No.

```
Public Sub Excel4SheetAndRangeTest()

   Dim dbs As Database
   Dim rstSales As Recordset
   Dim qdfNumOrders As QueryDef
   Dim intNumRecords As Integer

   ' Open the Microsoft Excel spreadsheet.
   ' Suppress headers (HDR=NO).
   Set dbs = OpenDatabase("C:\Samples", _
      False, False, "Excel 4.0;HDR=NO;")
   ' Create a Recordset for the entire worksheet
   ' WSSample.xls.
   Set rstSales = dbs.OpenRecordset("WSSample#XLS")
   ' Execute a MoveLast and count the records.
   rstSales.MoveLast
   intNumRecords = rstSales.RecordCount
   MsgBox "There are " & intNumRecords & _
      " rows in this worksheet."
   ' If headers aren't suppressed,
   ' intNumRecords would be one larger.
   rstSales.Close

   Set dbs = _
      OpenDatabase("C:\Samples\WSSample.xls", _
      False, False, "Excel 4.0;HDR=NO;")
   ' Create a Recordset for the named range
   ' "FirstTenRows" in the worksheet WSSample.xls.
   Set rstSales = dbs.OpenRecordset("FirstTenRows")
   ' Execute a MoveLast and count the records.
   rstSales.MoveLast
   intNumRecords = rstSales.RecordCount
   MsgBox "There are " & intNumRecords & _
      " rows in this named range."
   rstSales.Close
```

```
Set dbs = _
      OpenDatabase("C:\Samples\WSSample.xls", _
      False, False, "Excel 4.0;HDR=NO;")
   ' Create a Recordset for the unnamed range
   ' A1 through G5 in the worksheet WSSample.xls.
   Set rstSales = dbs.OpenRecordset("A1:G5")
   ' Execute a MoveLast and count the records.
   rstSales.MoveLast
   intNumRecords = rstSales.RecordCount
   MsgBox "There are " & intNumRecords & _
      " rows in this range."

   rstSales.Close
   dbs.Close

End Sub
```

This next example opens a worksheet in a Microsoft Excel 7.0 workbook from a Microsoft Jet database three times to demonstrate how to open a worksheet in a workbook, a named range in a workbook, and an unnamed range in a worksheet in the workbook. The code tests the ranges by counting the number of records that appear in the Recordset.

Note Because the formats of Microsoft Excel 5.0 and Microsoft Excel 7.0 are essentially the same, use the database type Excel 5.0 in the connection string for both Microsoft Excel 5.0 and Microsoft Excel 7.0. The specifier Excel 7.0 isn't valid.

```
Public Sub Excel7SheetAndRangeTest()

   Dim dbs As Database
   Dim rstSales As Recordset
   Dim qdfNumOrders As QueryDef
   Dim intNumRecords As Integer

   ' Open WBSample.xls.
   Set dbs = OpenDatabase _
      ("C:\Samples\WBSample.xls", _
      False, False, "Excel 5.0;HDR=NO;")
   ' Create a Recordset for the worksheet
   ' SampleSheet in the workbook WBSample.xls.
   ' The dollar sign character following the
   ' sheet name tells the IISAM that
   ' the code is referencing the entire sheet.
   Set rstSales = dbs.OpenRecordset("SampleSheet$")
   ' Execute a MoveLast and count the records.
   rstSales.MoveLast
   intNumRecords = rstSales.RecordCount
   MsgBox "There are " & intNumRecords & _
      " rows in this worksheet."
   rstSales.Close
```

```
    Set dbs = OpenDatabase _
       ("C:\Samples\WBSample.xls", _
       False, False, "Excel 5.0;HDR=NO;")
    ' Create a Recordset for the named range
    ' "SecondTenRows" in the workbook WBSample.xls.
    Set rstSales = dbs.OpenRecordset("SecondTenRows")
    ' Execute a MoveLast and count the records.
    rstSales.MoveLast
    intNumRecords = rstSales.RecordCount
    MsgBox "There are " & intNumRecords & _
       " rows in this named range."
    rstSales.Close

    Set dbs = OpenDatabase _
       ("C:\Samples\WBSample.xls", _
       False, False, "Excel 5.0;HDR=NO;")
    ' Create a Recordset for the range
    ' A1 through G5 in the worksheet
    ' SampleSheet in the workbook WBSample.xls.

    Set rstSales =dbs.OpenRecordset _
       ("SampleSheet$A1:G5")
    ' Execute a MoveLast and count the records.
    rstSales.MoveLast
    intNumRecords = rstSales.RecordCount
    MsgBox "There are " & intNumRecords & _
       " rows in this range."

    rstSales.Close
    dbs.Close

End Sub
```

Note Microsoft Excel 8.0 supports the ability to create files that contain formatting information for both Excel 7.0 and Excel 8.0. This enables the creation of files that can be opened by either Excel 7.0 and Excel 8.0 without having to convert the file format. You can access these files only in read-only mode when you use the Microsoft Excel IISAM.

Working with Lotus 1-2-3 Worksheets

The Microsoft Jet Lotus IISAM driver supports the following versions of the Lotus 1-2-3 worksheet product line: Lotus WKS, Lotus WK1, Lotus WK3, and Lotus WK4. Specific versions of Lotus worksheets allow different amounts of access to the data when they are linked or opened through the Lotus IISAM:

- Lotus .wks and .wk4 files allow read-only access to the data. You can't add, delete, or modify rows in worksheets created with these versions.
- Lotus .wk1 and .wk3 files allow read and insert access to the data. You can read all rows and add new rows, but you can't delete or modify existing rows in worksheets created with these versions.

Specifying Sheets and Ranges in Lotus Worksheet Files

You can specify a subset of the available data when you open a Lotus 1-2-3 single- or multiple-sheet worksheet file. In single-sheet files, you can open the entire sheet, a named range of cells, or an unnamed range of cells. In a multiple-sheet file, you can open a single sheet, a named range anywhere in the multiple-sheet file, or an unnamed range in a single sheet. The following table lists the conventions for the DATABASE and *source* arguments you must supply to access each of these objects.

To access this object	In this version of Lotus	Use this syntax
Entire sheet in a single-sheet file	WKS and WK1	Use the DATABASE parameter to specify the fully qualified network or directory path to the worksheet file; use the *source* argument to specify the worksheet as *filename*#**wks** or *filename*#**wk1**, where *filename* is the name of the worksheet.
Entire sheet in a multiple-sheet file	WK3 and WK4	Use the DATABASE parameter to specify the fully qualified network or directory path to the multiple-sheet file, including the multiple-sheet file name; use the *source* argument to specify the worksheet as *sheetname*:, where *sheetname* is the name of the worksheet. **Important** You must follow the worksheet name with a colon (:).
Named range of cells in a single- or multiple-sheet file	WKS, WK1, WK3, and WK4	Use the DATABASE parameter to specify the fully qualified network or directory path to the single- or multiple-sheet file, including the file name; use the *source* argument to specify the named range as *NamedRange*, where *NamedRange* is the name you assigned to the range in Lotus 1-2-3. **Important** You must name the range in Lotus 1-2-3 before attempting to open or link it.

To access this object	In this version of Lotus	Use this syntax
Unnamed range of cells in a single-sheet file	WKS and WK1	Use the DATABASE parameter to specify the fully-qualified network or directory path to the worksheet file, including the worksheet file name; use the *source* argument to specify the range as **A1..Z256**. Replace **A1..Z256** with the range of cells that you want to access.
Unnamed range of cells in a multiple-sheet file	WK3 and WK4	Use the DATABASE parameter to specify the fully qualified network or directory path to the multiple-sheet file, including the file name; use the *source* argument to specify the sheet you want to link or open as *sheetname*: and the range as **A1..Z256**. For example, to access cells A1 through Z256 in worksheet NetSales, you would use the following in the *source* argument: `NetSales:A1..Z256`.

Note You can't specify a value in a range that exceeds the maximum number of rows, columns, or sheets for the worksheet. See your Lotus 1-2-3 documentation for these values.

Linking Lotus Spreadsheets

To link a Lotus spreadsheet to your Microsoft Jet database, use the OpenDatabase method to open the local Microsoft Jet database, create a TableDef object, and set the Connect and SourceTableName properties of the TableDef object to indicate the path and name, respectively, of the Lotus spreadsheet that you want to link.

The following example links an external Lotus spreadsheet and then display a message box as confirmation:

```
Public Sub LinkLotusSheet()

   Dim dbsJet As Database
   Dim tdfThirdQtr As TableDef

   ' Open a Jet database.
   Set dbsJet = OpenDatabase("Sales.mdb")
   ' Create a TableDef object.
   Set tdfThirdQtr = dbsJet.CreateTableDef _
      ("Linked Lotus Table")
```

```
    ' Link the Lotus range Third Quarter in the
    ' spreadsheet C:\Lotus\Sales.wk3.
    tdfThirdQtr.Connect = "LOTUS WK3;" _
       & "Database=C:\Lotus\Sales.wk3"
    tdfThirdQtr.SourceTableName = "THIRDQTR"

    ' Append a TableDef object to create a link.
    dbsJet.TableDefs.Append tdfThirdQtr

    ' Display a confirmation message.
    MsgBox "Finished linking " _
       & tdfThirdQtr.SourceTableName & ".", 0

End Sub
```

Opening Lotus Spreadsheets

To open a Lotus spreadsheet directly, use the OpenDatabase method to indicate the path, exclusive access value, read-only value, and data source name.

The following example opens an external Lotus spreadsheet and then creates a Recordset object from the named range THIRDQTR:

```
Public Sub OpenLotusSheet()

    Dim dbsLotus As Database
    Dim rstThirdQtr As Recordset

    ' Open an external Lotus spreadsheet table.
    Set dbsLotus = OpenDatabase _
       ("C:\Lotus\Examples\Sales.wk3", False, _
       False, "LOTUS WK3;")

    ' Create a Recordset object from the Sales
    ' worksheet range "THIRDQTR".
    Set rstThirdQtr = dbsLotus. _
       OpenRecordset("THIRDQTR")

End Sub
```

Working with Text Files

You can use the Microsoft Jet Text IISAM to link and open character-delimited and fixed-length text files. Commas, tabs, or user-defined delimiters are valid in the source file.

When specifying connection information for text files, use the following specifications.

For	Use
Source database type	Text
Database name	The full path to the directory containing the text file you intend to access.
Source table name	The name of the text file, including the extension. If you don't specify an extension, the default .txt extension is used.

Microsoft Jet recognizes null values in character-delimited files by the presence of two consecutive delimiting characters. Microsoft Jet recognizes null values in fixed-length files by the absence of data (spaces) in the data column.

Microsoft Jet determines the format of the text file by reading the file directly or by using a schema information file. The schema information file, which is always named Schema.ini and always kept in the same directory as the text data source, provides the IISAM with information about the general format of the file, the column name and data type information, and a number of other data characteristics. A Schema.ini file is always required for accessing fixed-length data; you should use a Schema.ini file when your text table contains DateTime, Currency, or Decimal data or any time you want more control over the handling of the data in the table.

Note Microsoft Jet doesn't support multiuser access to text files. When you open a text file through Microsoft Jet, you have exclusive access to the file.

The following table lists the few limitations to the size of text tables and objects.

Item	Maximum size per text file
Field	255
Field name	64 characters
Field width	32,766 characters
Record size	65,000 bytes

Understanding Schema.ini Files

Schema.ini files provide schema information about the records in a text file. Each Schema.ini entry specifies one of five characteristics of the table:

- The text file name.
- The file format.
- The field names, widths, and types.
- The character set.
- Special data type conversions.

The following sections discuss these characteristics.

Specifying the File Name

The first entry in Schema.ini is always the name of the text source file enclosed in square brackets. The following example illustrates the entry for the file Sample.txt:

```
[Sample.txt]
```

Specifying the File Format

The Format option in Schema.ini specifies the format of the text file. The Text IISAM can read the format automatically from most character-delimited files. You can use any single character as a delimiter in the file except the double quotation mark ("). The Format setting in Schema.ini overrides the setting in the Windows Registry on a file-by-file basis. The following table lists the valid values for the Format option.

Format specifier	Table format
TabDelimited	Fields in the file are delimited by tabs.
CSVDelimited	Fields in the file are delimited by commas (comma-separated values).
Delimited(*)	Fields in the file are delimited by asterisks. You can substitute any character for the asterisk except the double quotation mark.
FixedLength	Fields in the file are of a fixed-length.

For example, to specify a comma-delimited format, you would add the following line to Schema.ini:

```
Format=CSVDelimited
```

Specifying the Fields

You can specify field names in a character-delimited text file in two ways:

- Include the field names in the first row of the table and set ColNameHeader to True.
- Specify each column by number and designate the column name and data type.

You must specify each column by number and designate the column name, data type, and width for fixed-length files.

Note The ColNameHeader setting in Schema.ini overrides the FirstRowHasNames setting in the Windows Registry on a file-by-file basis.

You can also instruct Microsoft Jet to determine the data types of the fields. Use the MaxScanRows option to indicate how many rows Microsoft Jet should scan when determining the column types. If you set MaxScanRows to 0, Microsoft Jet scans the entire file. The MaxScanRows setting in Schema.ini overrides the setting in the Windows Registry on a file-by-file basis.

The following entry indicates that Microsoft Jet should use the data in the first row of the table to determine field names and should examine the entire file to determine the data types used:

```
ColNameHeader=True
MaxScanRows=0
```

The next entry designates fields in a table by using the column number (**Col***n*) option, which is optional for character-delimited files and required for fixed-length files. The example shows the Schema.ini entries for two fields, a 10-character CustomerNumber text field and a 30-character CustomerName text field:

```
Col1=CustomerNumber Text Width 10
Col2=CustomerName Text Width 30
```

The syntax of Col*n* is:

Col*n*=*ColumnName type* **[Width** *#***]**

The following table describes each part of the Col*n* entry.

Parameter	Description
ColumnName	The text name of the column. If the column name contains embedded spaces, you must enclose it in double quotation marks.
Type	Data types are: Microsoft Jet data types Bit Byte Short Long Currency Single Double DateTime Text Memo ODBC data types Char (same as Text) Float (same as Double) Integer (same as Short) LongChar (same as Memo) Date *date format*

Parameter	Description
Width	The literal string value `Width`. Indicates that the following number designates the width of the column (optional for character-delimited files, required for fixed-length files).
#	The integer value that designates the width of the column (required if Width is specified).

Selecting a Character Set

You can select from two character sets: ANSI and OEM. The following example shows the Schema.ini entry for an OEM character set. The CharacterSet setting in Schema.ini overrides the setting in the Windows Registry on a file-by-file basis. The following example shows the Schema.ini entry that sets the character set to ANSI:

```
CharacterSet=ANSI
```

Specifying Data Type Formats and Conversions

The Schema.ini file contains a number of options that you can use to specify how data is converted or displayed when read by Microsoft Jet. The following table lists each of these options.

Option	Description
DateTimeFormat	Can be set to a format string indicating dates and times. You should specify this entry if all date/time fields in the import/export are handled with the same format. All of the Microsoft Jet formats except A.M. and P.M. are supported. In the absence of a format string, the Windows Control Panel short date picture and time options are used.
DecimalSymbol	Can be set to any single character that is used to separate the integer from the fractional part of a number.
NumberDigits	Indicates the number of decimal digits in the fractional portion of a number.
NumberLeadingZeros	Specifies whether a decimal value less than 1 and greater than –1 should contain leading zeros; this value can either be False (no leading zeros) or True.
CurrencySymbol	Indicates the currency symbol to be used for currency values in the text file. Examples include the dollar sign ($) and Dm.

Option	Description
CurrencyPosFormat	Can be set to any of the following values: • Currency symbol prefix with no separation ($1). • Currency symbol suffix with no separation (1$). • Currency symbol prefix with one character separation ($ 1). • Currency symbol suffix with one character separation (1 $).
CurrencyDigits	Specifies the number of digits used for the fractional part of a currency amount.
CurrencyNegFormat	Can be one of the following values: • ($1) • –$1 • $–1 • $1– • (1$) • –1$ • 1–$ • 1$– • –1 $ • –$ 1 • 1 $– • $ 1– • $ –1 • 1– $ • ($ 1) • (1 $) This example shows the dollar sign, but you should replace it with the appropriate CurrencySymbol value in the actual program.
CurrencyThousandSymbol	Indicates the single-character symbol to be used for separating currency values in the text file by thousands.
CurrencyDecimalSymbol	Can be set to any single character that is used to separate the whole from the fractional part of a currency amount.

Note If you omit an entry, the default value in the Windows Control Panel is used.

Examples of Schema.ini Files

Schema.ini contains the specifics of a text data source: how the text file is formatted, how it's read at import time, and the default export format for files. The following example shows the layout for a fixed-width file, Filename.txt:

```
[Filename.txt]
ColNameHeader=False
Format=FixedLength
MaxScanRows=25
CharacterSet=OEM
Col1=columnname Char Width 24
Col2=columnname2 Date Width 9
Col3=columnname7 Float Width 10
Col4=columnname8 Integer Width 10
Col5=columnname9 LongChar Width 10
```

Similarly, the format for a delimited file is specified as:

```
[Delimit.txt]
ColNameHeader=True
Format=Delimited(!)
MaxScanRows=0
CharacterSet=OEM
Col1=username Text
Col2=dateofbirth DateTime
```

If you are exporting data into a delimited text file, specify the format for that file as well.

```
[Export: My Special Export]
ColNameHeader=True
Format=TabDelimited
MaxScanRows=25
CharacterSet=OEM
DateTimeFormat=dd.mm.yy.hh.mm.ss
CurrencySymbol=Dm
CurrencyPosFormat=3
CurrencyDigits=2
CurrencyNegFormat=9
CurrencyThousandSymbol=.
CurrencyDecimalSymbol=,
DecimalSymbol=,
NumberDigits=2
NumberLeadingZeros=TRUE
```

The My Special Export example refers to a specific export option; you can specify any variation of export options at connect time. This last example also corresponds to a data source name (DSN) that can be optionally passed at connect time. All three format sections can be in the same .ini file.

Linking Text Files

The procedures for linking a comma-delimited file are similar to those for linking other database tables. If you want Microsoft Jet to use the information in the Schema.ini file, you must place the data file and the schema initialization file in the same directory.

The following example links a comma-delimited text file:

```
' This example links a comma-delimited text file
' to a Microsoft Jet database and tests the link
' by computing the sum of the extended prices.
Public Sub LinkCSVTextFile()

   Dim dbsJet As Database
   Dim tdfSales As TableDef
   Dim qdfNumOrders As QueryDef
   Dim rstTotalSales As Recordset

   ' Open the Microsoft Jet database.
   Set dbsJet = _
      OpenDatabase("C:\Jet_Samp.mdb")

   ' Create a Table for the text table.
   Set tdfSales = dbsJet.CreateTableDef _
      ("Central Division Sales")

   ' Provide a source database type and database name
   ' with the Connect property.
   tdfSales.Connect = _
      "TEXT;DATABASE=C:text_csv"

   ' Provide the name of the text table with the
   ' SourceTableName property.
   tdfSales.SourceTableName = "Sample.txt"

   ' Append the TableDef to create the link.
   dbsJet.TableDefs.Append tdfSales

   ' Run a simple query to ensure the data
   ' is accessible.
   Set qdfNumOrders = dbsJet.CreateQueryDef _
       ("Total Central Sales")
   qdfNumOrders.SQL = "SELECT Sum([Extended Price])" _
      & "As Tally FROM [Central Division Sales];"
   Set rstTotalSales = qdfNumOrders.OpenRecordset()
   MsgBox "Total Sales = $" & rstTotalSales!Tally

   qdfNumOrders.Close
   rstTotalSales.Close
   dbsJet.Close

End Sub
```

Opening Text Files

To open a text file directly, use the OpenDatabase method to indicate the path, exclusive access value, read-only value, and data source name.

This example opens an external text file then creates a Recordset object from the table.

```
Public Sub OpenCSVTextFile()

   Dim dbsText As Database
   Dim rstAwards As Recordset

   ' Open an external text data file.
   Set dbsText = OpenDatabase _
      ("C:\Winword\Data", False, False, "TEXT;")

   ' Create a Recordset from the Awards table.
   Set rstAwards = dbsText.OpenRecordset("AWARDS")

End Sub
```

Working with HTML Files

The Hypertext Markup Language (HTML) is a standard for presenting information over the Internet. HTML files are text files that include the information that users will see, along with *tags* that specify formatting information about how the information will be presented. For example, you can specify tags for bold, italic, underline, indentation, and many other types of formatting. Internet browsers are able to interpret information within HTML files and display them as pages to users. This is how you can create and view pages on the World Wide Web.

In addition to supporting text and graphics, HTML also supports tabular information. Using HTML table data tags (<TD> and </TD>), you can embed one or more tables within an HTML file. You can use the Microsoft Jet HTML IISAM to open HTML files and access the tabular data in them.

Linking HTML Files

You can access a table within an HTML file by creating a link to the table within a Microsoft Jet database, or by opening the HTML file directly by using the OpenDatabase method and opening a Recordset on the table within the file. To specify the location of the file, you must know the Uniform Resource Locator (URL) address of the HTML file on the Internet. The following code links an HTML table to a Microsoft Jet database:

```
Public Sub LinkHTML()

   Dim dbsJet As Database
   Dim tdfHTML As TableDef

   Set dbsJet = OpenDatabase("C:\Jet_Samp.mdb")

   ' Create a TableDef object.
   Set tdfHTML = dbsJet.CreateTableDef _
      ("Linked HTML Table")

   ' Set connection information.
   tdfHTML.Connect = _
      "HTML Import;DATABASE=http:" _
      & "//www.usa.myserver.com/files/mypage.html"
   tdfHTML.SourceTableName = "Q1SalesData"

   ' Append a TableDef object to create a link.
   dbsJet.TableDefs.Append tdfHTML

End Sub
```

Opening HTML Files

To open an HTML file directly, use the OpenDatabase method to indicate the path, exclusive access value, read-only value, and data source name. The following example uses the OpenDatabase method to open an HTML file directly:

```
Public Sub OpenHTML()

   Dim dbsHTML As Database
   Dim rstSales As Recordset

   ' Open an external HTML file.
   Set dbsHTML = OpenDatabase _
      ("http://www.usa.myserver.com" _
      & "/files/mypage.html", _
      False, False, "HTML Import;")

   ' Create a Recordset object from
   ' the Q1SalesData table
   Set rstSales = dbsHTML.OpenRecordset("Q1SalesData")

End Sub
```

Note When you open an HTML file, the HTML IISAM uses the HTTP protocol to copy the source file from the Internet server to the workstation requesting the data. The HTML IISAM then accesses this local copy of the file, not the file on the server. The records you access through code are only a snapshot of the actual records in the file on the server. Changes made to the server copy of the file aren't available to your application until you reopen or relink the HTML file. The copy of the file on the Internet server is never modified.

The local copy of the HTML file is read-only. You can't modify tabular data in an HTML file. However, you can read tabular HTML information as a Recordset, save the contents of the Recordset in a new table in your local Microsoft Jet database, and then export the table as an HTML file. The export operation creates an HTML file that includes the minimum necessary HTML header information, plus the contents of the table.

You can also access an HTML file by specifying the FTP location of the file in the OpenDatabase method. For example:

```
Set dbs = OpenDatabase _
   ("ftp://www.usa.myserver.com" _
   & "/files/mypage.html", _
   False, False, "HTML Import;")
```

If you specify an FTP location, the target file on the FTP server is automatically copied into a new subdirectory of the temporary directory specified in the TEMP environment variable. The new subdirectory is named Net0000.tmp.

Table Naming Conventions

As the previous example shows, you must reference a table inside an HTML file by a name, for example, Q1SalesData. If a table in an HTML file has a caption, the caption is the table name. If a table doesn't have a caption, and it's the only table in the file, use the title of the HTML document to reference the table. If more than one table exists in the HTML file, and none of these tables has captions, you can reference them sequentially in code as "Table1", "Table2", and so on. The IISAM interprets these as "the first unnamed table in the file," "the second unnamed table in the file," and so on.

Column Names and Data Types

As the HTML IISAM reads the data in the HTML file, it chooses the data type for a column by interpreting the contents of the cells. For example, if most of the values in a column are numeric, the HTML IISAM interprets the data as Long or Double, depending on whether most of the numbers are integer or floating point values. Similarly, if the majority of the values in a column is text, the HTML IISAM interprets the data as Text (with a field size of 255).

Important In columns that contain a combination of data types, the HTML driver assigns Null to cells of the minority data type. For example, if a column of data in an HTML table contains 90 integers and 10 strings, the HTML IISAM will create a field with a Long data type and interpret the strings as Null. If you then use the OpenRecordset method on the HTML table, your code will encounter 90 integers and 10 Null values in that field.

If you don't want the HTML IISAM to choose data types, you can force columns of data to have specific data types by creating a Schema.ini file, which is kept in the same directory as the local HTML file from which data is being imported. The Schema.ini file contains information about each column of data that is imported using the HTML IISAM. The format of the Schema.ini file is similar to the Schema.ini file that is used to import data in text files. The only difference is that in the Schema.ini file for HTML files, the section headings indicate the table name, not the file name. For example, in the following HTML Schema.ini file, the table name is Sales.

```
[Sales]
ColNameHeader=False
Format=FixedLength
MaxScanRows=25
CharacterSet=OEM
Col1=columnname Char Width 24
Col2=columnname2 Date Width 9
Col3=columnname7 Float Width 10
Col4=columnname8 Integer Width 10
```

By default, the HTML IISAM assumes that the first row of the table doesn't contain field names. You can override this by using the `HDR=` keyword in the connection string. For example, to indicate that the first row of the table contains field names, include `"HDR=Yes"` in the connection string.

Although the HTML specification supports a table header tag (<TH>), the HTML IISAM doesn't assume that this tag indicates a field name. The HTML IISAM will treat all data within the <TH> and <\TH> tags as normal table data so you can embed <TH> tags within the table, not just place them in the first row.

For More Information For more information about creating a Schema.ini file, see "Working with Text Files" earlier in this chapter.

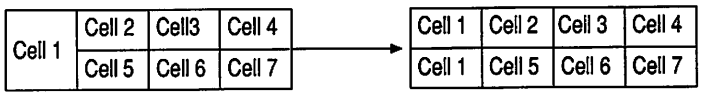
Complex Table Considerations

Some tables within HTML files are simple combinations of rows and columns, with each cell having a unique identity. However, you can define more complex tables with HTML. For example, a single cell can span more than one column, more than one row, or a combination of rows and columns. You can use this for embedded column and row headings that span multiple cells. You can also embed lists within tables and even other tables within tables. The HTML IISAM can handle these complex tables. The following sections describe how it handles each complex table feature.

Column and Row Span

When an HTML table includes cells that span multiple columns, the HTML IISAM divides the spanning cell into multiple cells, each with the same value. The following diagram shows how the HTML IISAM interprets a cell spanning multiple columns.

Similarly, for a cell spanning multiple rows, the spanning cell is subdivided like this.



When a cell spans multiple rows and columns, the spanning cell is divided like this.

Handling Embedded Images and Hyperlinks

You can embed graphics within table cells with HTML. References to graphics are identified through the HTML <IMG SRC ... > tag. If a cell contains just this tag and no other text, the IISAM interprets the cell as empty, because the HTML IISAM can't import graphics. If the cell contains a graphic and some other text, the IISAM will import the text, but not the graphic. Similarly, if the cell contains an ALT tag to indicate text that should be displayed instead of a graphic, the text after the ALT tag is imported.

When the reference to the graphic is a hyperlink tag of the form <A HREF=".....">TEXT<\A>, the text between the <A> tag and the <\A> tag will be imported. The hyperlink text is always imported as a Memo field.

If the cell contains only a hyperlink and no other embedded text, the HTML IISAM imports the caption and URL associated with the hyperlink.

If the cell contains more than just a single link, the link is ignored but any other embedded text is imported.

Handling Embedded Lists and Tables

Some HTML tables may have lists embedded within one or more table cells. In this case, the HTML IISAM inserts a carriage return and line feed after each list item. It doesn't create separate cells for the list items. If the list of items separated by carriage returns is less than 255 characters, the IISAM creates a Text field; if the list is longer than 255 characters, the IISAM creates a Memo field.

A "stand-alone" list that isn't embedded in a table is interpreted by the IISAM as a single-column table.

If a table is embedded within another table, the tables are treated as separate tables. The cell occupied by the embedded table is set to Null in the enclosing table, and the embedded table is treated as a separate table from which you can create a separate Recordset.

Note The HTML IISAM doesn't support formatting tags that can be placed within the <TD> and <\TD> tags. These include tags such as ID, LANG, BORDER, WIDTH, FLOW, ALIGN, and so on. If the HTML IISAM encounters these tags, it ignores them.

Registry Settings

Two Windows Registry settings pertain to the HTML IISAM:

- HTML Import
- HTML Export

Use the HTML Import Registry setting when reading, importing, or linking more than one table per file. The default value for the OneTablePerFile Registry setting is False because there can be many tables within a single HTML file.

The HTML Export OneTablePerFile Registry setting is always True because the IISAM will export only one table per file. This is different from all other IISAMs, because they all import or export the same number of tables per file.

Note Because the HTML IISAM operates in a very similar way to the Text IISAM, Registry entries for importing text also apply to importing HTML files.

Working with Microsoft Exchange Mailboxes

Microsoft Exchange is an information sharing and messaging system that enables users to access and exchange information over a network. In Microsoft Exchange, information is stored within mailboxes that consist of a series of folders. Folders can contain other folders or a series of messages received from other users. For example, a user can create a folder to contain all personal messages in his or her mailbox, and a folder to contain all business messages.

You can access messages within a Microsoft Exchange mailbox or public folder by using the Microsoft Exchange installable ISAM. This means that you can open a Microsoft Exchange mailbox as a Database object, and you can open a folder inside that mailbox as a Recordset object. You can access the individual messages inside a folder as records, each of which has a series of fields. Examples of message fields are To, From, Subject, Received, and Size.

For example, you can use the Microsoft Exchange IISAM to write a program that accesses a user's mailbox and displays a report showing all messages in all folders that are over one year old. Or you could write a program that calculates and displays statistics on how frequently a certain public folder has new messages sent to it.

Additionally, once you open a Microsoft Exchange mailbox as a Database object, you can enumerate the folders in the mailbox by enumerating the TableDefs collection on the database (each TableDef represents a folder). Similarly, for each folder, you can enumerate the Fields collection of the TableDef object representing the folder to determine which fields are defined within the folder.

Exchange Connection Strings

To access messages in a Microsoft Exchange mailbox or public folder, you must use the connection string keyword `MAPILEVEL=`. This keyword specifies the path to the Exchange mailbox that is the parent of the folder you want to access. For example, to access the folder named Barbara in the Important folder in the People folder in the mailbox "Mailbox - Dave Jones (Exchange)", use the following connection string:

```
"Exchange 4.0;MAPILEVEL=Mailbox - Dave Jones (Exchange)|People\Important;"
```

You can then use the OpenRecordset method and specify the folder named "Barbara". You can specify any valid mailbox name to the left of the pipe symbol, but the mailbox name must be exactly as it appears in the left pane of the Microsoft Exchange application window. Any spaces and capitalization in the name must be preserved.

In most cases, Microsoft Exchange users must supply a profile name and password before gaining access to mailboxes. When you use the Microsoft Exchange IISAM, you supply the profile name and password with the `PROFILE=` and `PWD=` keywords in the connection string. For example:

```
"Exchange 4.0;MAPILEVEL=Mailbox - Dave Jones
(Exchange)|People\Important;PROFILE=DaveJ;PWD=Cisco"
```

For More Information For more information about connection strings, see "Specifying Connection Information" earlier in this chapter, or see "Connect Property" in the *Language Reference* in Books Online.

Linking Microsoft Exchange Folders

You can access a folder within a Microsoft Exchange mailbox by creating a link to the folder within a Microsoft Jet database, or by opening the folder file directly by using the OpenDatabase method and opening a Recordset on the folder within the mailbox. When you link a folder, the folder is treated as a table. The following code links a Microsoft Exchange folder to a Microsoft Jet database so that you can access it as if it were a table.

```
Public Sub LinkExchangeFolder()

   Dim dbsJet As Database
   Dim tdfExchange As TableDef
   Dim str As String

   Set dbsJet = OpenDatabase("c:\Jet_Samp.mdb")

   str = "Exchange 4.0;MAPILEVEL=" _
      & "Mailbox - Dave Jones " _
      & "(Exchange)|People\Important;"

   ' Create a TableDef object.
   Set tdfExchange = dbsJet.CreateTableDef _
      ("Linked Exchange Folder")

   ' Set connection information.
   tdfExchange.Connect = str
   tdfExchange.SourceTableName = "Barbara"

   ' Append a TableDef object to create a link.
   dbsJet.TableDefs.Append tdfExchange

End Sub
```

Opening Microsoft Exchange Folders

To open a Microsoft Exchange folder directly, use the OpenDatabase method to indicate the path, exclusive access value, read-only value, and data source name. The following code example uses the OpenDatabase method to open a Microsoft Exchange folder named "Barbara" and count the number of messages received from a user named Jim Harris:

```
Public Sub OpenExchangeFolder()

   Dim dbsExchange As Database, intCount As Integer
   Dim rst As Recordset, str As String

   str = "Exchange 4.0;MAPILEVEL=" _
      & "Dave Jones (Exchange)|People\Important;"

   Set dbsExchange = OpenDatabase _
      ("C:\Data\Temp.mdb", 0, 0, str)

   Set rst = dbsExchange.OpenRecordset("Barbara")
```

```
rst.MoveFirst
   While Not rst.EOF
         If rst!From = "Jim Harris" Then
            intCount = intCount + 1
         End If
         rst.MoveNext
   Wend

   rst.Close
   dbsExchange.Close

End Sub
```

Note When using the OpenDatabase method to open a folder in a Microsoft Exchange mailbox, you must supply the path and name of an existing Microsoft Jet database as the first argument. The Microsoft Exchange IISAM creates system tables in this database to store information about the structure of the Microsoft Exchange folders being accessed, including all the new fields you've appended to these folders. It is usually best to create a new, empty Microsoft Jet database for this purpose and use this same database each time you want to access data in a folder. If you delete this database and use a new one, all the information about fields you added to the folders will be lost, and you must recreate them.

Creating and Deleting Folders

You can use DAO to create new folders within a Microsoft Exchange mailbox by using the CreateTableDef method on the Database object representing the Microsoft Exchange mailbox. The new folder is created at the current level in the Microsoft Exchange mailbox.

You can also delete folders within a Microsoft Exchange mailbox by using the Delete method on the TableDefs collection of the Database object representing the Microsoft Exchange mailbox.

Note If you are accessing a Microsoft Exchange folder from Microsoft Access, remember that Microsoft Access tables are limited to 64 characters and can't contain certain characters. In contrast, there is no restriction on Microsoft Exchange folder names.

Constraints

In Microsoft Exchange, some folders contain only other folders and no messages. You can't access this type of folder by using the Microsoft Exchange IISAM. It is only possible to access folders containing messages.

You can only access existing messages in a Microsoft Exchange mailbox in read-only mode. You can't update information in a Microsoft Exchange mailbox, but you can append new records (messages) to an existing folder. Additionally, you can't use the Seek method because the Microsoft Exchange IISAM doesn't support indexing.

You can add new fields to existing folders or to newly created folders in a Microsoft Exchange mailbox, but you can't delete fields. You can rename folders but you can't rename fields.

Maintaining Links

Links to external tables are convenient because they store all of the connection information you need to access the table in the future. Connection information for linked tables resides in the TableDef object that represents that table in your Microsoft Jet database. If you move or delete the external table, you must update the corresponding TableDef object to reflect this change.

Updating Links to Tables That Have Moved

If you move a table that you previously linked, you need to refresh the link before you can access the table again. To refresh a link, you must reset the Connect property of the TableDef object to point to the table's new location. Then, use the RefreshLink method of the TableDef object to update the link information.

The following procedure shows how to reset the location for a FoxPro table that has been moved to another directory. The FoxPro table LostLink.dbf was originally linked to Jet_Samp.mdb from the \Samples directory. The table has since been moved to the \Samples\LostLink directory. When you try to open the linked table, Microsoft Jet reports that it couldn't find the object "LostLink."

1. Try to open the table LostLink.dbf. Microsoft Jet displays the message "Couldn't find object 'LostLink'."
2. Run the following code:

```
Public Sub RefreshLostFoxProLink()

   Dim dbs As Database
   Dim qdfNumOrders As QueryDef
   Dim rstTotalSales As Recordset

   ' Open the Microsoft Jet database.
   Set dbs = OpenDatabase("C:\Samples\Jet_Samp.mdb")
   dbs!LostLink.Connect = "FoxPro 3.0" & _
      ";DATABASE=C:\Samples\LostLink"
   dbs.TableDefs!LostLink.RefreshLink
```

```
    ' Run a simple query to ensure the data
    ' is accessible.
    Set qdfNumOrders = dbs.CreateQueryDef _
       ("Test Access to Refreshed Link")
    qdfNumOrders.SQL = "SELECT Sum (Ext_price) As" _
       & "Tally FROM LostLink;"
    Set rstTotalSales = qdfNumOrders.OpenRecordset()
    MsgBox "Total Sales = $" & rstTotalSales!Tally

    rstTotalSales.Close
    dbs.Close

End Sub
```

3. Try to open the table LostLink.dbf. This should succeed.

Deleting Links to External Tables

You can also delete the TableDef object associated with an external table. When you delete a TableDef that defines a linked table from your Microsoft Jet database, the external table is unaffected.

Performance and Troubleshooting Tips

When you link or open external tables in a Microsoft Jet database, it's easy to forget that you're accessing the data through an additional service layer (the IISAM). When you consider the amount of overhead involved in maintaining a connection, transferring and translating data, and maintaining a remote index in a foreign format, you can see why certain Microsoft Jet operations are significantly slower when you execute them against an external data source. The first part of this section discusses one way to dramatically increase performance when you only need to read data in a single pass. The remainder of the chapter focuses on two common problems: connections and temporary space.

Using Forward-Only Recordset Objects

If your application only needs to read through records in an external table in a single pass, you can dramatically increase the performance of your application by using forward-only–type Recordset objects. Although this option limits the direction of travel through the Recordset and restricts the use of a few other methods, it offers

perhaps the greatest performance improvement possible when used with very large external tables. The following methods and properties are unavailable when you open Recordset objects of type dbOpenForwardOnly:

Bookmark property

Clone method

FindFirst method

FindLast method

FindNext method

Move method (with any value other than 1)

MoveFirst method

MoveLast method

MovePrevious method

OpenRecordset method

For More Information For more information about the dbOpenForwardOnly option, see Chapter 3, "Working With Records and Fields," or see "OpenRecordset Method" in the *Language Reference* in Books Online.

Resolving Connection Problems

If your application can't connect to the external data source, you can check for the following:

- Make sure you can access the external table through the file system. If the table is stored on a network share, for example, check that you have sufficient privileges to access that share.
- Check that the external table is accessible through its native application. For example, if you're trying to open a Microsoft Excel worksheet, check that you can open the worksheet in Microsoft Excel.
- Check the code that establishes the connection to the external table carefully to make sure that it follows the guidelines in "Specifying Connection Information" earlier in this chapter.
- If you were able to connect to the external table through your application before but you can't now, check that the table is still in its original location. If the table is linked in your Microsoft Jet database and it has been moved, see "Maintaining Links" earlier in this chapter.
- If you receive the message "Unable to find installable ISAM," check the source database type value in the table in "Specifying Source Database Types," earlier in this chapter. (Remember that Excel 7.0 isn't a valid source database type; use Excel 5.0 instead.)

- If the source database type matches one of the values in the table in "Specifying Source Database Types," check that the IISAM driver resides in the directory specified in the Windows Registry.
- If you are supplying parameters to a case-sensitive data source, check that the parameters are of the correct form and case.
- Make sure that the system on which the external data source resides has sufficient disk space.

Understanding Temporary Space Needs

Microsoft Jet creates temporary indexes on your local computer when you query a database. This occurs even when the database is remotely located and in a different file format. These indexes can be as small as a few kilobytes or as large as several megabytes, depending on the data source.

You should make sure users of your application have sufficient resources to accommodate these temporary space needs. All users should have a defined value for the TEMP variable and enough disk space to store the indexes that Microsoft Jet may create.

Note If users lack the disk space necessary to store temporary indexes, your application may behave unpredictably.

CHAPTER 8

Creating Multiuser Applications

If you're developing an application for multiple users on a network, you want to make sure that your application helps users share data effectively. This chapter covers issues you should consider when creating a multiuser application. It explores locking strategies that control multiuser access to data and discusses ways your application can handle locking conflicts in a multiuser environment.

Note This chapter provides information on data sources connected through Microsoft Jet, not through ODBCDirect. With ODBCDirect, your application can directly interact with Open Database Connectivity (ODBC) data sources without going through the Microsoft Jet database engine. In some situations, this can significantly improve performance, but it has some limitations. For more information on ODBCDirect, see Chapter 9, "Developing Client/Server Applications."

Managing Shared Databases

To make your application available to multiple users, do one of the following:

- Place the entire application on a network server and allow multiple users to open the database. With this strategy, users share all the database objects that make up your application, including the tables it uses to store data.
- Split your application into two databases: a back-end database that contains your application's tables, and a front-end database that contains your application's other objects — its queries, forms, reports, macros, and modules. The front-end database can also manage temporary objects. (Most multiuser applications perform some type of data access that is temporary in nature, such as creating tables with transient data.) Then, place the back-end database on a network server and distribute the front-end database to your users. Base the objects in your front-end database on linked tables from the back-end database.

The advantages of the two-database approach become obvious in multiuser environments. Storing the front-end database on the user's workstation eliminates contention for temporary objects, because your application doesn't have to create a temporary table that may conflict with another user. Second, storing static objects at the local level minimizes the amount of network traffic that occurs while your application is running.

Location of Components

In most scenarios, the very act of implementing a database system in a multiuser environment incurs a substantial performance penalty because accessing files over a network is typically slower than accessing files on a local computer.

To minimize network traffic, you can store some of your application's components on the local workstation. For this discussion, components are grouped into two categories:

- *Static components* are the parts of your application that don't change often. The dynamic-link libraries (.dll) and executable files (.exe) that make up the host program calling Microsoft Jet—such as Visual Basic, Microsoft Access, or Microsoft Excel—are examples of static components. Components in your application, such as forms, reports, and program code, are also static components.
- *Active components* are the files that contain the actual data that your users access in a multiuser environment. These can include Microsoft Jet databases (.mdb), external databases, such as Microsoft FoxPro database files, or ODBC databases stored on database servers.

Considerations for Static Components

Because static components don't change often, they are ideal candidates for workstation storage. For optimal performance, install the Microsoft Jet components and the host application on the local workstation along with your application's static objects. However, you do have to update objects on local workstations when the application's design changes.

If your user workstations don't have sufficient disk space, or you would like greater control over changes to static components, you can store static components on the network. This has the obvious effect of increasing network traffic and reducing application performance, but it minimizes administrative issues. If possible, it's usually better to store static components on the users' workstations.

Self-Updating Applications

You should keep in mind that administrative concerns arise when you move components to the workstation level. For example, when you need to modify a static object in your application, such as a table's structure or a program's code, you must have mechanisms in place for updating all affected components on each individual workstation.

To minimize this problem, check for the latest versions of your components when your application starts. For example, your application's startup code can compare the modification dates of static objects in the local database to dates in a central database located on the server. When the code finds a newer date on a server object, it automatically imports the latest version of the object, keeping the local database up to date.

You can also use the Microsoft Jet replication features to synchronize objects. For example, you can create replicas of the database and store them at the workstation level. You can then use replication features to automatically synchronize objects.

For More Information For more information on replication, see Chapter 6, "Database Replication."

Considerations for Active Components

As is the case with static components, the location of your application's active components can be a major factor in determining your application's overall performance

For example, assume an application's data file is located on a network server, and the application has to search for a record on a table that has no indexes. In this scenario, Microsoft Jet must perform a sequential scan of the entire table to find a match. This means that the entire table must be sent from the server to the workstation's copy of the engine. If the table is large, this can be a lengthy operation.

The problem is further compounded because the user running the search is waiting a long time for a response, and the high volume of network traffic is, in turn, slowing down other users. To alleviate this situation, add indexes for the columns on which your application searches. However, having indexes may increase network traffic and reduce concurrency and efficient performance.

In another example, assume that an application uses a set of tables containing lookup data that represents postal codes or lists of part numbers, and it doesn't change often. Moving this type of data to the local workstation eliminates network access that may occur hundreds of times during the application's typical execution.

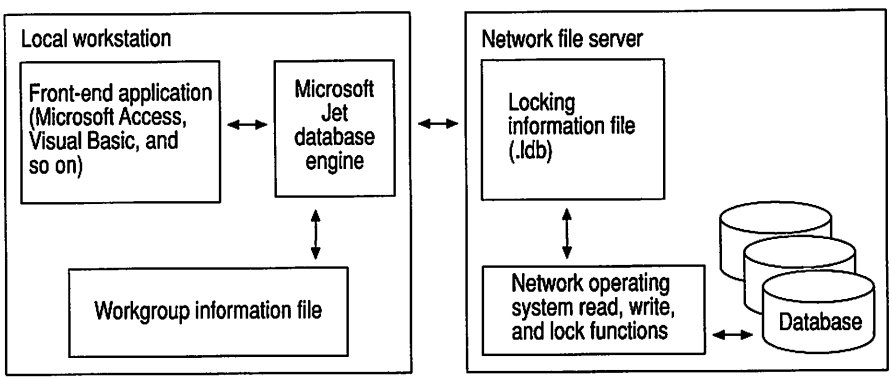
The Microsoft Jet Multiuser Model

A Microsoft Jet multiuser database application consists of several important components. A copy of Microsoft Jet runs on each user's workstation, and if you use the two-database approach, static components will usually reside on each user's computer, and active components will reside on a network server. In addition, your multiuser application must have:

- A workgroup information file, which resides either centrally on the network server, or on each user's workstation.
- A locking information file (.ldb) for each database, which resides in the same folder as the database.

The following diagram illustrates how these components are related.

Figure 6.1 The Microsoft Jet multiuser model


Local workstation
Front-end application (Microsoft Access, Visual Basic, and so on)
Microsoft Jet database engine
Workgroup information file
Network file server
Locking information file (.ldb)
Network operating system read, write, and lock functions
Database

The Workgroup Information File

The workgroup information file is a Microsoft Jet database that stores user, group, and password information, as well as information on various options the user has set. In Visual Basic, you can creade this file using the Visdata application, available on the **Add-Ins** menus.

The location of the workgroup information file in multiuser environments is important. You can either place it on each local workstation or share it on a network server. However, if it's stored locally, you must take the steps necessary to update it when security settings change.

Note If a database resides on a user's local workstation, Microsoft Jet requires the availability of some form of operating-system locking. The network services built into Microsoft Windows 95 and Windows NT® handle this.

The Locking Information File

The locking information (.ldb) file stores information about any records that are currently locked in the database.

If the locking information file doesn't exist when the database is opened, the Microsoft Jet database engine creates it. Microsoft Jet creates one locking information file for every Microsoft Jet database file that is opened in shared mode. It gives the file the same name as the database that was opened, but with an .ldb file name extension. For example, Northwind.mdb has an associated locking information file called Northwind.ldb. The locking information file is stored in the same folder as the database. Microsoft Jet deletes the .ldb file when the database is closed. In a multiuser setting, this is when all users have exited the application.

Multiuser Settings

You can use a variety of settings to control how Microsoft Jet behaves in a multiuser environment. The Windows Registry contains these settings, and you can change them with the Registry Editor (regedit.exe for Windows 95 or regedt32.exe for Windows NT).

You can directly update the Windows Registry to modify engine settings for applications that use Microsoft Jet. The benefit to this approach is that you make settings directly to Microsoft Jet. All applications that use the engine will then use these settings.

For More Information For more information on pessimistic and optimistic locking, see "Using Page Locking" later in this chapter.

Supported Networks

Microsoft Jet supports the following file-server and peer-to-peer networks:

- Microsoft Windows NT and Windows 95 networks
- Banyan VINES 5.52
- DECnet 4.1
- LANtastic 5.0
- Novell NetWare 3.x and 4.x
- OS/2 LAN Manager, 2.1 and 2.2

Although Microsoft Jet supports these networks equally, the performance of peer-to-peer networks doesn't generally meet the requirements of a multiuser database system. Most successful applications that perform database access are implemented on dedicated file-server network operating systems.

Locking Shared Data in Recordsets

When working in a single-user environment, your application doesn't encounter conflicting requests for resources. Because only one user is accessing data, you can reasonably assume that the needed data will always be exclusively available. However, after you move into a multiuser environment, the only reasonable assumption is that conflicts will occur among users accessing data. Microsoft Jet manages these conflicts through the locking services that it provides to the application. These services ensure that only one user can simultaneously access data at a given time.

To protect shared data, you can lock data while a user edits it. When data is locked, any number of users can read it, but only one user can make changes to it.

To understand locking, you must be familiar with the levels at which you can "lock" data. With Microsoft Jet, you can lock data at three different levels. These vary from the most restrictive to the least restrictive.

- *Exclusive mode* prevents all other users from accessing the database. This is the most restrictive mode.
- *Recordset locking* locks a Recordset object's underlying tables with either table-read locks, table-write locks, or both.
- *Page locking* locks the 2048 byte (2K) page containing the data that is being edited. This is the least restrictive mode.

To determine how to lock objects in your application, you must decide the level of concurrency that you need. For example, if you want the objects to be available as often as possible, a high-concurrency strategy would dictate that you use page locking, which is the least restrictive level. However, if your application requires guaranteed access to most or all of the data in the database, you may opt for exclusive mode. This ensures that your application has exclusive access to the database.

Note that the three levels aren't mutually exclusive. For example, if you have an order entry system, you could use page locking to control locks on an orders table to enable the highest concurrency among order takers. You could then use recordset locking at the end of the day to lock a summary table to update it with summary data. Finally, you could use exclusive mode nightly to compact the database.

Comparing Shared and Exclusive Modes

As previously mentioned, exclusive mode is the most restrictive way you can open a database: It prevents all other users from opening the database. This is useful for performing administrative or bulk changes to the database, such as repair or compact operations, or for making structural changes to the database's schema.

When you access a database in a single-user environment, you typically open it in exclusive mode. This may provide better performance because Microsoft Jet doesn't have to lock and unlock objects or refresh its cache. When you deploy your application in a multiuser environment, you can't use exclusive mode; you must open the database in shared mode.

You may want to prevent users from opening a shared database in exclusive mode. This is particularly true if they use applications, such as Microsoft Access, that allow users to open databases exclusively through the user interface. For example, to prevent a user from opening the database in exclusive mode, set the user's Open Exclusive permission to No on a database. You probably should allow administrative users the option of opening the database exclusively.

Opening a Database in Shared Mode

If you want to implement recordset locking, you must open your database in shared mode. When you open a database in shared mode, multiple users can simultaneously access the database and Microsoft Jet handles conflicts between users trying to edit the same record. To open a database in shared mode, use the OpenDatabase method to open the database, specifying a value of False for the *options* argument.

The following code opens a database in either shared or exclusive mode. If `blnExclusive` is True, the database is opened in exclusive mode; otherwise, the database is opened in shared mode. This code also turns off error trapping with the On Error Resume Next statement, and it uses inline error handling to detect errors.

```
Function OpenDatabaseX(dbs As Database, strDBPath As _
   String, blnExclusive As Boolean) As Integer

   ' Turn off error trapping.
   On Error Resume Next

   ' Open the database located at strDBPath. If
   ' blnExclusive is True, open the database in
   ' exclusive mode; otherwise, open it in shared
   ' mode.
   Set dbs = OpenDatabase(strDBPath, blnExclusive)

   Select Case Err
      ' You should declare constants, such as SUCCESS
      ' and ERR_NOPERMISSION, so that they are
      ' available to this procedure.
      Case 0: OpenDatabaseX = SUCCESS
      Case 3033: OpenDatabaseX = ERR_NOPERMISSION
      Case 3343: OpenDatabaseX = ERR_BADDATABASE
      Case 3044: OpenDatabaseX = ERR_INVALIDPATH
      Case 3024: OpenDatabaseX = ERR_FILENOTFOUND
      Case Else: OpenDatabaseX = FAILED
   End Select
```

```
    ' Set Err back to zero so that the error
    ' generated here isn't sent back to the caller.
    Err = 0

End Function
```

To use this function, declare a Database object variable with the Dim statement, and then pass this object variable to the OpenDatabaseX function, along with the full path and name of the database (`strDBPath`). The code calling OpenDatabaseX should examine the return value of OpenDatabaseX to see if any errors occurred, and it should take action depending on the error.

Using Read-Only Mode

Read-only mode is a modified form of shared mode. When you open a database in read-only mode, you cannot change data or objects in the database. However, other users can change data, and you shouldn't confuse this mode with opening the file in read-only mode from an operating system level. The following code fragment opens a database in exclusive, read-only mode:

```
Set dbsOrdEntry = OpenDatabase _
   ("OrdEntry.mdb", True, True)
```

To open a database in shared, read-only mode, use the following code:

```
Set dbsOrdEntry = OpenDatabase _
   ("OrdEntry.mdb", False, True)
```

Note Opening a database in read-only mode doesn't prevent shared table-type locks or read locks; therefore, opening a database in read-only mode doesn't prevent locking conflicts.

Using Recordset Locking

With recordset locking, you can control how the Recordset object's underlying tables are locked. You can specify read-only or write-only access, or both. Recordset locking applies only to table- and dynaset-type Recordset objects. Page locking doesn't apply to snapshot-type and forward-only–type Recordset objects, which are inherently read-only objects. However, Microsoft Jet places shared table-read and shared table-write locks.

Opening a Recordset

After you open the database in shared mode, you can specify various options in the *options* argument of the OpenRecordset method to implement recordset locking.

▶ To open a Recordset with locking enabled

1 Open the Recordset object's database in shared mode.

2 Open the Recordset with the OpenRecordset method and set the *options* argument to the locking mode that you want.

3 Close the Recordset object when you're finished with it to release any recordset-level locks.

For example, the following code opens a table exclusively by combining the dbDenyWrite and dbDenyRead *options* constants. If the function is successful, no other users can access the table until the Recordset variable is explicitly or implicitly closed. If another user has the table opened exclusively, or an unanticipated error occurs, the function returns the user-defined global constant FAILED.

```
Function OpenTableExclusive(dbs As Database, rst As _
   Recordset, strTable As String) As Integer

   Set rst = dbs.OpenRecordset(strTable, _
      dbOpenTable, dbDenyRead + dbDenyWrite)

   Select Case Err
      Case 0: OpenTableExclusive = SUCCESS
      Case Else: OpenTableExclusive = FAILED
   End Select

   Err = 0

End Function
```

Note If you open a Recordset object without specifying a value for the *options* argument, Microsoft Jet uses page locking. This opens the Recordset in shared mode, and it doesn't prevent other users from accessing the data in the Recordset. However, it does lock the data that's being edited in the current page.

For More Information For more information on opening a Recordset, see "Creating a Recordset Variable" in Chapter 3, "Working with Records and Fields."

Checking for Errors When Opening Recordsets

As with opening databases exclusively, setting locks on Recordset objects can cause errors if the lock fails. You can handle recordset locking problems with a generic procedure that takes into account the most common multiuser errors.

The most common error in locking Recordset objects occurs when another user has the Recordset open in a way that prevents you from obtaining the lock that you want. This is identified as error 3262, "Couldn't lock table <name>; currently in use by user <name> on machine <name>." This error occurs when you attempt to use the OpenRecordset method on an object that cannot be locked. Usually this happens when another user has the same table or tables locked in a way that prevents your lock. To respond to the error, wait a short period of time and retry the operation.

Using Page Locking

Page locking locks only the page containing the record that is currently being edited. Microsoft Jet accesses and locks data in pages of 2048 bytes. When you use page locking, which is the default locking level for Recordset objects, other users can read data from the locked page, but they cannot change it.

To control page locking, you specify the locking mode by using the LockEdits property. In the parts of your application that lock data at the page level, you must determine which locking mode you're going to use. The locking mode, either pessimistic locking or optimistic locking, determines how Microsoft Jet actually locks the data.

Pessimistic Locking

With *pessimistic locking,* the engine locks the page containing the record currently being edited as soon as the Edit method is issued, and it doesn't release the lock until the changes to the record are explicitly committed or canceled. Pessimistic locking is the default locking mode for Recordset objects.

Note Transactions change the behavior that determines when a lock is released.

The primary advantage of pessimistic locking is that after you have obtained a lock, you know that you won't encounter any locking conflicts as long as the record is locked. Additionally, pessimistic locking is the only way to guarantee that your application reads the most current data, because one user can't change a record after another user has started to edit it.

The disadvantage of pessimistic locking is that the entire page that contains the record is locked for the duration of the procedure that edits and locks the record. If you're giving the user access to data through a user interface, another problem arises when the user starts to edit a record, locks it, and leaves the computer for a period of time. This causes not only the record that the user is editing to be locked, but possibly other records that reside within the locked page.

Note You can set the type of locking for a Recordset in the *lockedits* argument of the OpenRecordset method. You can also use the LockEdits property to set the type of locking after you open a Recordset. If you set the LockEdits property to True, pessimistic locking is enabled. If you set LockEdits to False, optimistic locking is enabled.

For More Information For more information on the OpenRecordset method, see Chapter 3, "Working with Records and Fields."

Optimistic Locking

With optimistic locking, the engine locks the page only when you try to post the record changes with the Update method. Because the lock occurs only when your application attempts to post changes, you minimize the time the lock is in place; this is the main advantage of optimistic locking.

The disadvantage of optimistic locking is that when a user begins to edit a record, you can't be sure that the update will succeed. An update that relies on optimistic locking fails if another user changes a record that the first user is editing. Imagine a scenario where users Mike and Maya are editing the same record. Mike starts editing the Acme customer record with optimistic locking. Because Mike's optimistically locked record isn't actually locked, nothing prevents Maya from attempting to edit the same record. Maya starts editing the same Acme record. Unfortunately, Maya has no idea that Mike is already editing the record, and also doesn't have the most up-to-date view of the data. When Mike attempts to save his changes, he receives an error because Maya is also editing the record.

▶ **To use optimistic locking in your code**

1. Open a table- or dynaset-type Recordset on the data that you want to edit.
2. Move to a record.
3. Enable optimistic locking by setting the LockEdits property of the Recordset to False.
4. Use the Edit method to allow edits to the record (the record isn't yet locked).
5. Make changes to the record.
6. Use the Update method to post the record changes (this attempts to lock the record).
7. Check to see if the Update method succeeded. If it didn't, try again.

Optimistic locking turns into pessimistic locking when you use transactions. Because a transaction holds a write lock until you commit the transaction, pessimistic locking occurs even if the LockEdits property is False.

Also, it's possible for the Update method to fail in optimistic locking. Just because the Edit method hasn't placed a write lock, it doesn't mean that the Update method hasn't placed a write lock. In other words, one user could have a Recordset open with

pessimistic locking, and a second user's update to the same data fails even if the second user is using optimistic locking.

The following code implements pessimistic locking. It updates the number of units in stock for a given product in the Products table.

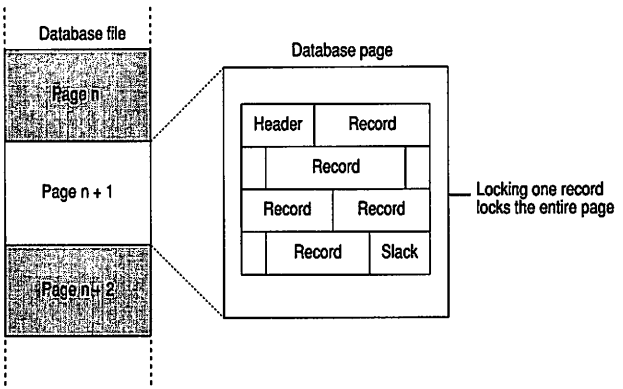
```
Function UpdateUnitsInStock(strProduct As String, _
   intUnitsInStock As Integer, intMaxTries As Integer)

   Dim dbs As Database
   Dim rstProducts As Recordset
   Dim blnError As Boolean
   Dim intCount As Integer
   Dim intLockCount As Integer
   Dim intChoice As Integer
   Dim intRndCount As Integer
   Dim i As Integer

   On Error GoTo ErrorHandler

   ' Open the database specified in the user-defined
   ' constant, STR_DBPATH, in shared mode.
   Set dbs = OpenDatabase(STR_DBPATH)

   ' Open the table for editing.
   Set rstProducts = dbs.OpenRecordset("Products", _
      dbOpenDynaset)

   With rstProducts
      ' Set the locking type to pessimistic. Setting
      ' LockEdits to False would use optimistic
      ' locking.
      .LockEdits = True

      .FindFirst "ProductName = " & Chr(34) & _
         strProduct & Chr(34)

      If .NoMatch Then
         UpdateUnitsInStock = ERR_NOMATCH
         GoTo CleanExit
      End If

      ' Attempt to edit the record. If a lock error
      ' occurs, the error handler attempts to resolve
      ' it. Because this procedure uses pessimistic
      ' locking, errors occur when you try to edit a
      ' record. If it used optimistic locking, errors
      ' would occur when you update a record.
      .Edit
      ![UnitsInStock] = intUnitsInStock
      .Update

   End With
```

```
CleanExit:
   rstProducts.Close
   dbs.Close
   Exit Function

ErrorHandler:
   Select Case Err

      Case 3197
         ' Data in the Recordset has changed since it
         ' was opened. Try to edit the record again.
         ' This will automatically refresh the
         ' Recordset to display the most recent data.
         Resume

      Case 3260' The record is locked.
         intLockCount = intLockCount + 1
         ' Tried to get the lock twice already.
         ' Let the user cancel or retry.
         If intLockCount > 2 Then
            intChoice = MsgBox(Err.Description _
               & " Retry?", vbYesNo + vbQuestion)
            If intChoice = vbYes Then
               intLockCount = 1
            Else
               UpdateUnitsInStock = ERR_RECORDLOCKED
               Resume CleanExit
            End If
         End If

         ' Yield to Windows.
         DoEvents
         ' Delay a short random interval, making it
         ' longer each time the lock fails.
         intRndCount = intLockCount ^ 2 _
            * Int(Rnd * 3000 + 1000)
         For i = 1 To intRndCount: Next i

         Resume' Try the edit again.

      Case Else ' Unanticipated error.
         MsgBox "Error " & Err & ": " & Error, _
            vbOKOnly
         UpdateUnitsInStock = FAILED
         Resume CleanExit

   End Select

End Function
```

Locking Records

One of the most common questions developers ask when they start their first multiuser Microsoft Jet application is: How do I lock a record? The short answer is: You don't. You have seen that Microsoft Jet reads, writes, and locks data a page at a time, not a record at a time. Depending on the size of the record, the page may contain more than one record. The problem that arises is that when you lock a record, you lock all the records on that page. This may cause a problem in high-concurrency applications that must provide unhindered access to specific records. You can see how this works in Figure 6.2.

Figure 6.2 Page locking



You can use various strategies to work around this design.

One strategy is to add columns to the table by using the Text data type until the record length is greater than 1024 bytes. Using the Text data type isn't recommended for this implementation because it's a variable-length data type, and you must explicitly pad the fields with data to achieve a fixed-length format.

This technique may cause performance degradations because the size of the database will increase if every record occupies 2K of disk space.

The recommended way is to use the SQL DDL CHAR data type, which is a fixed-length data type. With this method, you don't have to pad the field with data as you do with the methods described in the preceding paragraphs. Therefore, using the CHAR data type is the easiest and only guaranteed way to force this technique.

Another strategy is to use optimistic locking wherever possible. Although optimistic locking doesn't eliminate page locking, it does minimize the amount of time that a record is locked and therefore lowers the possibility that an unwanted record will also be locked.

Finally, if none of these approaches is acceptable, you may want to consider either moving your application to a client/server environment that uses a server that supports record locking, or implementing your own custom locking scheme as described in the following section.

Implementing a Custom Locking Scheme

Sometimes page locking isn't appropriate, and optimistic locking may not be a suitable workaround. In this case, you may want to consider your own custom locking scheme. Your code can bypass Microsoft Jet locking and identify when to lock and unlock a record.

You can use a lock table to identify when a record is locked. The lock table stores the key value of the record, the lock status (locked or unlocked), and the name of the user who has the record locked.

The implementation of a custom locking scheme requires a great deal of design, implementation, and testing time. In many cases, it cannot duplicate functionality that is built into Microsoft Jet. For example, even if you implement single-record locking, it would be very difficult to handle data in a Recordset that's based on more than one table, because you would have to identify all the tables that contain records that have to be locked. Custom locking schemes are most attractive when they affect only a few tables and aren't based on a data model with complex joins and relationships.

Checking for Errors with Page Locking

If you're using page locking, your code must check to see if an attempted lock succeeded. As with the previous examples, you should disable error handling, attempt the operation that will initiate a lock, check for errors, and finally, re-enable error handling.

Most of the page-locking multiuser errors that your code encounters will be one of the following three errors. These aren't the only errors — just the most commons ones.

Error code and text	Cause and suggested response
3186 Couldn't save; currently locked by user *<name>* on machine *<name>*.	This error occurs when a user attempts to update a page that contains a read lock placed by another user. To handle this error, wait for a short period of time, and then try to save the record again. Optionally, you can inform users of the problem and allow them to indicate whether or not they want to retry the operation.
3197 The Microsoft Jet database engine stopped the process because you and another user are attempting to change the same data at the same time.	This error occurs when you use the Edit method or the Update method and another user has changed the current record since you opened the Recordset or last read data from the record. If this error occurs when you use the Edit method, you may want to refresh the user's view of the data with the current data and then attempt the Edit method a second time. If this error occurs when you use the Update method, then you're using optimistic locking and the record has changed since you used the Edit method. Inform the user that someone else has changed the data. You may want to display the current data and give the user the choice of whether to overwrite the other user's changes or cancel edits.
3260 Couldn't update; currently locked by user *<name>* on machine *<name>*.	This error occurs when you use the AddNew or Edit method and the page containing the current record is locked. This error also occurs when you use the Update method to save a record on a locked page. This situation can occur when the user is trying to save a new record or when you're using optimistic locking and another user locks the page. To handle this error, wait for a short interval, and then try to save the record again. Optionally, you can inform users of the problem and allow them to indicate whether or not they want to retry the operation.

Testing a Record's Locking Status

You may want to test a record to see if it's locked. The following procedure tests whether the current record is on a locked page. To do this, it sets locking on the Recordset to pessimistic and tries to edit the record. If the record is locked, an error is generated.

```
Function RecordLocked(rst As Recordset) As Boolean

   Dim blnLock As Boolean

   On Error GoTo ErrorHandler

   ' Save the current value of LockEdits property.
   blnLock = rst.LockEdits

   ' Set pessimistic locking.
   rst.LockEdits = True

   ' Try to edit the record. This generates
   ' error 3197 if the record is locked.
   rst.Edit
   RecordLocked = False
   rst.CancelUpdate

   ' Restore the original value of LockEdits property.
   rst.LockEdits = blnLock

   Exit Function

ErrorHandler:
   Select Case Err
      Case 3197:
         Resume Next
      Case Else
         RecordLocked = True
         Resume Next
         Exit Function
   End Select

End Function
```

Note Although you can use the previous code to test a record to see if it is locked, there is no guarantee that the record will remain in the state reported by the function. If you want to edit a record, don't call a separate function to see if the record is locked. Instead, try to edit the record and handle any errors that may arise, as was shown earlier in this chapter in the UpdateUnitsInStock function procedure.

Other Multiuser Design Issues

Up to this point, this chapter has discussed locking as the main issue with multiuser applications. However, it isn't the only issue. When designing an application for multiuser access, you should keep in mind several additional factors, as discussed in the rest of this chapter.

Refreshing Data Access Objects Collections

In a multiuser setting, users can modify collections when they add new tables or queries to the database, or delete existing objects. In this case, DAO doesn't automatically keep track of changes to collections.

Take the example of the following code, which displays all of the TableDef objects in the current database:

```
Sub ShowAllTables(dbs As Database)

   Dim tdfTemp As TableDef

   For Each tdfTemp In dbs.TableDefs
      Debug.Print tdfTemp.Name
   Next tdfTemp

End Sub
```

When you run this code, the code takes a snapshot of the TableDefs collection as it exists at that time. It iterates through each TableDef object and displays its name. Assume that you have to run this code twice. The first time you run the code, it displays ten table names. But before you can run it again, another user on the network, who also has the database open, adds a new table. If you run the code again, it will still display ten table names. This is because DAO doesn't automatically know about changes made to the collections by other users. To force DAO to update the collection with the most recent changes, use the Refresh method.

To change the previous code so that is always returns the most current list of TableDef objects, add a Refresh method:Sub BetterShowAllTables(dbs As Database)

```
   Dim tdfTemp As TableDef

   dbs.TableDefs.Refresh
   For Each tdfTemp In dbs.TableDefs
      Debug.Print tdfTemp.Name
   Next tdfTemp

End Sub
```

Note Use the Refresh method only if necessary. If a collection contains a large number of objects, refreshing the collection may take a long time.

Requerying Data

If your multiuser application presents data to the user in a visual form, such as a window, you may want to update the user's view with the most current data. Your application must explicitly requery data to get the most current view of other users' changes.

To obtain the most current view of data in a Recordset, you must first determine if the Recordset supports the Requery method. If the Recordset object's Restartable property value is True, you can use the Requery method to refresh the Recordset object's contents.

If the Recordset doesn't support the Requery method, you must open the Recordset again with the OpenRecordset method. In general, re-opening a Recordset is slower than using the Requery method.

This following code tries to requery a specified Recordset. If the requery succeeds the function returns SUCCESS. If the code couldn't reqery the Recordset, the function tries to re-open the Recordset. If it can't re-open the Recordset, the function returns FAILED. The `dbs` parameter is the database in which the Recordset `rst` was first opened.

```
Function RequeryRecordset(dbs As Database, rst As _
   Recordset) As Integer

   On Error Resume Next

   ' Can the Recordset be requeried?
   If rst.Restartable = False Then
      RequeryRecordset = FAILED
      Exit Function
   End If
```

```
    rst.Requery        ' Requery the Recordset.

    Select Case Err' Handle any errors.
       Case 0
          RequeryRecordset = SUCCESS
          Exit Function
       Case Else
          Err = 0
          ' Re-open the Recordset.
          ' Use rst.Name to get the value of the SQL
          ' string, table, or QueryDef on which the
          ' Recordset was originally based.
          Set rst = dbs.OpenRecordset(rst.Name, _
             rst.Type)
          Select Case Err
             Case 0
                RequeryRecordset = SUCCESS
                Exit Function
             Case Else
                ' Don't return error to caller.
                Err = 0
                RequeryRecordset = FAILED
                Exit Function
          End Select
       End Select

End Function
```

In general, most Recordset objects are restartable (or can be requeried). The exceptions are Recordset objects based on pass-through queries and on crosstab queries without fixed column widths. You can't requery these types of Recordset objects, and you must reopen them to get the most current state of the data.

Using Transactions in a Multiuser Environment

You can use Microsoft Jet transactions to group updates in atomic units that can be committed or rolled back as a whole. Because transactions buffer updates to a temporary file instead of to real tables, they also have useful effects in multiuser environments. The most common use of transactions in multiuser environments is to make sure users don't see an incomplete view of shared data while it's being changed.

For example, assume your application is running code that is updating data, and another user is simultaneously running a report on that data. If your updates aren't wrapped in a transaction, the user running the report could receive inconsistent data. By wrapping your updates in a transaction, you make sure the other user's view of the shared data isn't affected by your code until your code has finished executing.

Be careful of keeping a transaction open too long. All locks that result from your edits with the transaction are kept in place until the transaction is committed or rolled back. This can have harmful effects on the multiuser concurrency of your application.

For More Information For more information on transactions, see "Managing Transactions" in Chapter 3, "Working with Records and Fields."

Identifying Users

In your multiuser application, you may want to programmatically identify the user currently logged on to the system. This is useful for administrative functions, such as storing the user's name with edited records to create an audit trail. Several methods are available to achieve this functionality. One involves Microsoft Jet security and another requires you to maintain your own user information.

By implementing a secure database, you force the user to log on to your application with a predefined user name and password. The user name is then available to your application through the UserName property of the Workspace object.

The following code writes the user's name and the current date and time to a specified Recordset:

```
Function WriteAuditTrail(rst As Recordset) As Integer

   On Error GoTo ErrorHandler

   ' Edit the current record in the Recordset.
   ' Assume that the Recordset (rst) includes the
   ' fields UserLastModified and DateTimeLastModified.
   rst.Edit
   rst!UserLastModified = Workspaces(0).UserName
   rst!DateTimeLastModified = Now
   rst.Update

ErrorHandler:
   Select Case Err
      Case 0
         WriteAuditTrail = SUCCESS
         Exit Function
      Case Else
         MsgBox "Error " & Err & ": " & Error, _
            vbOKOnly
         WriteAuditTrail = Err
         Exit Function
   End Select

End Function
```

If you don't want to implement security, but still require user name functionality, you can have your application prompt the user for a name and password at startup and store those values in code variables or a temporary table. This way, you still have access to the user's name, but you don't have to implement a secured database. However, this kind of security provides only a thin layer of protection for your application's data. A knowledgeable person could easily gain access to the data under this type of security.

Network Hardware Considerations

You can improve performance in a multiuser setting by reducing hardware bottlenecks on your LAN. You might want to try the following:

- Segment the LAN into subnets. Typically, a server has only one network interface card (NIC) with many nodes connected to it. To reduce this bottleneck, add multiple NICs to the server and spread the nodes across those segments. This is usually the least expensive and most significant performance gain because it substantially reduces collisions on the wire and gives each station a larger slice of bandwidth to communicate with the server.
- Add a better NIC to the server. Practically all high-end systems now come with PCI bus slots. Use PCI NICs for significantly faster throughput while keeping the CPU utilization lower. Also, many PCI NICs offer a multiple-ported design, thus allowing you to create multiple segments from one PCI NIC. For example, a server with two PCI slots can use two four-port PCI NICs. This creates eight segments that go to 36 nodes to reduce collisions on the wire and increase the bandwidth that each node has when communicating with the server. This allows you to focus on performance issues that are software-related instead of hardware-related.

  Another option with PCI NICs is the 100 MB Ethernet standard. While requiring 100 MB hubs to accompany the 100 MB NICs is expensive, it reduces performance issues of going over the wire and it uses existing cables.
- Add faster disk drives and host adapters to the server. Again, moving to PCI- or EISA-based host adapters can have a significant impact on performance. One way to eliminate this bottleneck is to configure the server to use an EISA SCSI RAID controller that uses eight 1 GB SCSI drives. This increases performance because the EISA card handles some processing itself and reduces CPU processing. Because the RAID configuration allows reads and writes to be spread across the eight disk drives, it reduces contention issues when trying to access a drive. This is probably the most expensive solution, but it is a significant performance improvement over a one ISA card–one disk solution.

CHAPTER 9

Developing Client/Server Applications

The Microsoft Jet database engine and DAO (Data Access Objects) both contain a number of features that enable you to create powerful client/server applications. This chapter introduces you to the concepts of client/server computing and shows you how to create client/server applications using DAO with Microsoft Visual Basic.

This chapter also explains how to use the ODBCDirect extensions to DAO. ODBCDirect enables you to access the ODBC applications programming interface (API) through the DAO object model.

Client/Server Concepts

In Chapter 8, "Creating Multiuser Applications", you saw how Microsoft Jet and DAO can be used to build applications in a file-server environment. Recall that in a file-server database architecture, there is no database engine on the server — the server merely provides access to network files. The simplicity of the file-server architecture makes it ideal for successfully implementing small to moderate-sized applications.

However, there are certain applications whose needs go beyond the capabilities of file-server database systems. For example, once an application exceeds a certain threshold of concurrent transactions, or needs to take advantage of advanced features such as triggers, a client/server architecture may be better. Additionally, there are many performance, security, data-integrity and administration issues that arise which file-server database systems cannot accommodate. In this situation, you may want to consider using a client/server architecture.

Comparing File-Server and Client/Server

In a file-server database configuration, the Microsoft Jet database engine resides on each user's workstation and the data file containing all the data resides on a server. When your application needs data, the database engine on the workstation processes the query and then issues a request for specific data on the network drive. Because all requested data in the data file must be sent from the server to the workstation, this can generate a significant amount of network traffic.

In a client/server configuration, both the database engine and the data needed by the application reside on the network server. Instead of running a copy of the database engine on each workstation, the database engine runs only on one machine — the server. The workstation simply sends a high-level request in the form of an SQL statement to the server, which retrieves the specific data and sends it back to the client.

How it Works

To better compare how client/server systems differ from file-server systems, consider the following Visual Basic code :

```
Dim dbs As Database
Dim rstTest As Recordset
Dim strName As String

Set dbs = DBEngine(0).OpenDatabase ("mydb.mdb")
Set rstTest = dbs.OpenRecordset ("mytable")
strName = rstTest!LastName
```

In a file-server database application, the following operations are performed:

1. Microsoft Jet opens the file mydb.mdb, a very inexpensive operation in terms of time and resources.
2. The Microsoft Jet DLL running on your workstation looks inside the database file for information about a table named MyTable. This results in several relatively inexpensive disk reads and some memory manipulations on your machine.
3. Microsoft Jet loads details about the MyTable table into your workstation memory, such as the number of fields, field data types, available indexes, and information about the position of the first data page in the database file.
4. The executing copy of Microsoft Jet on your workstation reads the actual data from the disk and makes it available to the application.

In a client/server system, these operations are performed as follows:

1. As in steps 1 and 2 in the previous list, Microsoft Jet opens the file mydb.mdb, retrieves some information from it, and then sets up some structures in memory on the local workstation.
2. When Microsoft Jet loads MyTable, it sees that it is a link to an ODBC database and tries to create a connection to the ODBC server. This is a relatively expensive operation which may take several seconds.
3. When the ODBC server receives a connection request, it sets aside memory and other resources and prepares for subsequent requests. It then replies with a connection.

4. Microsoft Jet issues a request to the server for the MyTable data requested in the DAO code.
5. The server returns the requested data to Microsoft Jet, which makes it available to the client application.

The Advantages of Client/Server

For many database applications, the client/server architecture offers several advantages, including the following:

- More reliable and robust operations because a single database server interacts with the data (instead of multiple copies of a file-share database).
- Better scalability due to database servers as opposed to file-share databases. If more users are added to the system, it is easier to scale the capabilities of the system to meet the additional requirements.
- Significantly improved performance of some operations, especially when the user workstations are low-end computers with slow processors and limited RAM. For example, executing a large query may be many times faster on a high-end server than it is on a typical workstation.
- Reduced network traffic due to the more efficient transmission of data. Only data that the application needs is transferred.
- Mission-critical features such as transaction logs, sophisticated backup capabilities, redundant disk arrays, and failure recovery tools.

Two Paths to ODBC Data

There are two distinct ways to access ODBC data sources with DAO:

- The Microsoft Jet database engine
- ODBCDirect

Microsoft Jet provides a broad range of functionality. It handles many of the details of establishing and managing connections, translating queries into a form acceptable to the server, and managing returned data.

In some cases, however, you may want to bypass the Microsoft Jet database engine and use DAO to make calls directly to the ODBC Driver Manager. This functionality is available through ODBCDirect.

Microsoft Jet and ODBCDirect each have their own advantages. You can use both simultaneously in your application, drawing on the strengths of each.

Accessing ODBC Data with Microsoft Jet

Microsoft Jet and ODBCDirect workspaces provide different but complementary functionality. You should use a Microsoft Jet workspace to access MDB files and ISAM data formats (such as text and spreadsheets). Also use it when you require Microsoft Jet-specific functionality to access ODBC data.

The following capabilities are supported by Microsoft Jet but not by ODBCDirect:

- Updatable joins — You can update data in Recordset objects based on multi-table joins.
- Support for linked tables — Microsoft Jet allows you to store persistent links to server data in a local Microsoft Jet database. Linking a table allows you to cache information about the table's structure, including field and index information, in your local database. The next time you need to access that table, the connection will be quicker because the structure information does not need to be retrieved from the server again.
- Support for the Find methods — Microsoft Jet Recordset objects support FindFirst, FindNext, FindPrevious, and FindLast methods.
- Partial failures of update queries — If you have a bulk-operation query, and it fails for some reason, the query will stop, allowing you to decide whether or not you want to commit the changes made up to the point of failure.
- User-defined properties — With Microsoft Jet, you can add persistent properties to existing objects. You can use these property objects to allow further definition of Microsoft Jet objects. For example, you can add a Description property to objects that allows you to assign descriptive text to each object.
- Crosstab queries — The Microsoft Jet engine supports the SQL TRANSFORM statement, which enables you to create crosstab queries that can be used to summarize data.
- Heterogeneous data access — You can access server data, native Microsoft Jet database (MDB) data, and external ISAM data such as FoxPro, Paradox, Btrieve and dBASE data. You can perform joins of tables in different data sources.
- Programmatic DDL — You must use a Microsoft Jet workspace to perform DDL operations using DAO. ODBCDirect does not provide a TableDef object, so you cannot create or modify tables. You can, however, perform DDL operations using ODBCDirect by executing SQL DDL statements.
- Form and control binding — If your application requires that forms or controls be bound to data in an ODBC data source, you must use Microsoft Jet. Data accessed within an ODBCDirect workspace cannot be bound to forms or controls because ODBCDirect does not support linked tables.

If you don't need these capabilities, you can use ODBCDirect workspaces exclusively. However, you can define both Microsoft Jet and ODBCDirect workspaces in your application at the same time. For example, in the same function, you can define a Microsoft Jet workspace to perform DDL operations using DAO and you can also define an ODBCDirect workspace to perform asynchronous queries.

Accessing ODBC Data with ODBCDirect

ODBCDirect gives you a more direct way to access server data by using the existing DAO object model directly on top of the ODBC API. ODBCDirect implements a thin code layer over the ODBC API that establishes connections, creates cursors, and executes complex procedures using minimal workstation resources. ODBCDirect offers the following advantages:

- Direct Access — With ODBCDirect, your application can access ODBC data sources directly, making the server responsible for all query processing.
- Reduced resource requirements — Since you don't have to load the Microsoft Jet database engine, your application requires fewer resources at the workstation.
- Improved access to server-specific functionality — For example, for servers that support it, ODBCDirect allows you to specify where cursors are located — locally or on the server. In addition, to interact with stored procedures at the server, you can specify input values and check return values — both operations that are not possible when using Microsoft Jet.
- Asynchronous queries — You can execute a query and perform other operations without waiting for the query to finish. You can then check properties to keep track of the query's execution. This provides greater possibilities for your application in terms of concurrency and optimized performance.
- Batch Optimistic Updating — Enables you to cache Recordset changes locally and then submit these changes to the server in a single batch.
- More flexible stored procedure execution — With ODBCDirect, you can handle output parameters and return values from stored procedures.

Installing ODBC Drivers

Before you can access data on a server, you must install the appropriate Open Database Connectivity (ODBC) drivers. If you chose Typical installation when you installed Microsoft Access or Visual Basic, you don't have these drivers installed. To install the ODBC drivers, run Setup again, click the Custom/Complete button, and make sure you select the appropriate ODBC drivers. Additionally, for each remote database to which you want to connect, you must set up an ODBC data source using the ODBC Manager program found in your Control Panel folder.

For More Information For more information about using ODBC, search for "ODBC" in Books Online.

Note Some database server documentation refers to records as *rows* and to fields as *columns*. This chapter uses the terms *records* and *fields*.

Moving Your Data to a Server

The best approach to creating a client/server application is to create the tables that will contain the data on the server itself. However, because the Microsoft Jet database engine works nearly the same with tables from any source, you can also create the tables in a local Microsoft Access database, export them to a server and then link the remote tables in your local database. If you have a copy of Microsoft Access, this second technique may be more convenient — especially if your database design isn't complete — but moving the data to the server can cause some complications, including the following.

Indexes

Exporting a table to a server creates no remote indexes. If you need to update the table's data, you must create a unique index on it using a pass-through query or a separate server-based tool before linking the exported table. You'll probably want to define other indexes as well to improve query performance.

AutoNumber Fields

Most servers don't provide the automatic-numbering functionality of an AutoNumber field. To mimic an AutoNumber field, you have to either create an insert trigger on the server that provides the next key value or write a Visual Basic procedure to provide this functionality.

Default Values

Default field values aren't exported with a table, but since most servers support default values, you can create them directly on the exported table.

Validation Rules

You can either use a remote table to enforce your rules or you can define validation rules in Visual Basic code. Implementing validation rules in Visual Basic is generally easier, but not as reliable because these rules are only enforced by your application, not by the server itself. Also, if you're using a server-based trigger, be aware that many Visual Basic functions have no equivalent in most database servers (such as the DatePart or Format functions).

Relationships

The Jet database engine doesn't recognize enforced relationships or cascading relationships that may exist on your server. In other words, Jet won't enforce these relationships — it merely updates data according to the actions users take through the user-interface. If you attempt a data operation on the server that violates the server's referential integrity, the server generates an error through the ODBC interface. You can mimic these relationships by using server-based triggers or local event procedures. You can mimic enforced relationships by prohibiting operations that violate the relationship, and you can mimic cascading relationships by programmatically synchronizing keys when you update related tables.

Security

Security that you establish in your Jet database has no effect on security on the server. DAO isn't aware of server security, although it cannot violate that security. For example, if you're editing a remote table for which you don't have permission to insert data, DAO doesn't prevent you from attempting to insert a new record. However, when you try to save the new record, the server returns and error message and the record isn't inserted. To minimize these discrepancies, synchronize local and remote user permissions and passwords.

Table and Field Names

DAO provides great flexibility in naming tables and fields for native Jet databases. You can use almost any combination of letters, numbers and symbols, and names can contain up to 64 characters. Since most servers do not offer this degree of flexibility, each non-alphanumeric character in every name is converted to an underscore (_) when you export a Microsoft Jet table to a server. As a result, queries, forms, reports and other objects in your database may no longer work.

Properties

You can't change most field properties on linked tables (regardless of whether or not they are ODBC or ISAM data sources). Even for those that you can change (such as InputMask, Format, and Caption), you must manually copy their property settings from the local table to the linked remote table.

Visual Basic Code

If you export a table to a remote database and then link it, you may need to modify modules containing code that uses the table. Here are a few examples:

- When accessing ODBC data sources, you can't open Recordset objects with the dbOpenTable option and therefore you can't use the Index property or the Seek method on remote tables. You may therefore need to replace the dbOpenTable option in calls to the OpenRecordset method with dbOpenSnapshot or dbOpenDynaset.

- DAO properties and methods that change table structures are usually not supported on remote tables; for example, you can't add a field to the Fields collection of a linked remote table. You may need to change this code to SQL DDL statements that are executed by the server database.
- If your code opens dynaset-type or snapshot-type Recordset objects on large ODBC tables, you should consider filtering the data when you call the OpenRecordset method. In other words, instead of specifying the table name as the *source* parameter in the OpenRecordset method, use an SQL SELECT statement with a WHERE clause. For example, if you are opening the Employees table but your code is only going to search for Sales Representatives, you should use a `SELECT * FROM Employees WHERE title = 'Sales Representative'`.
- Selecting all fields from a table on a remote data source can be expensive, especially if Memo or Long Binary fields are involved. If your code only uses a subset of the available fields, consider changing the SELECT statement in your OpenRecordset method to select only needed fields. For example, `SELECT LastName, Phone FROM Employees`.

Accessing ODBC Data With Microsoft Jet

As described earlier in this chapter, Microsoft Jet and DAO provide substantial programming flexibility for accessing data in ODBC data sources. This section shows how to access ODBC data using Microsoft Jet and DAO.

Connecting to a Server

To gain access to a remote server, your application must establish a connection to the server across the network. In some cases, establishing the connection is handled automatically for you, in other cases, you establish connections explicitly as your application requires.

You can use any of the following methods to connect to a server using the Jet database engine.

- Link tables and views.
- Use pass-through queries to send SQL statements straight to the server.
- Directly access the server using DAO

Linking Tables and Views

The most efficient way to access ODBC data through Microsoft Jet is to link tables on the server rather than opening the database directly. You can then use queries in your application just as you would use them in a Microsoft Jet database.

Although opening a database directly may sound faster than linking, this is not the case. When you access data through Visual Basic code using the OpenDatabase method, information about the table (including its fields and indexes) must be retrieved every time the table is opened. When you link a table, this information is stored permanently in the local database. Because the information is cached in a local database, it takes less time to access the table.

It is important to note that since a linked table stores structural information about a table and its fields and indexes locally, changes to the structure of the table on the server are not automatically synchronized back to your local database. You must relink remote tables if you change the structure of the tables or their indexes on the server.

Linking SQL Views and Creating a Pseudo Index

If your server supports the creation of SQL views, you can link them to your local database. The Jet engine treats a linked view exactly like a linked table with no indexes. The processing defined in a view is always performed by the server, no matter what operations are run locally.

If your server allows you to update data using views, and you want to take advantage of this updatability in Visual Basic, you need to create a *pseudo index* on the linked view. This pseudo index tells your program which field or fields uniquely identify a record returned by the view. Your program can then create an updatable dynaset on the view, which forms and queries can use.

For example, suppose you link an SQL view named SeptOrdersView that returns a subset of records in the Orders table on the server, and you name the linked view SeptemberOrders. Because the OrderID field is still unique within the view, you would run a data-definition query (not a pass-through query) by executing the following SQL statement:

```
CREATE UNIQUE INDEX index1 ON SeptemberOrders (OrderID)
```

Because you ran a data-definition query and not a pass-through query, this doesn't create an index on the server, nor does it create a real index in your local database. It simply tells your program that the OrderID field is a unique key of the linked view.

Accessing ODBC Data with DAO

There are several different ways to access data within an ODBC data source. If you're just getting started with client/server computing, the following steps offer a "quick start" to accessing ODBC data using DAO:

1. Use the ODBC data source manager in the Windows Control Panel to set up your ODBC data source and assign it a Data Source Name (DSN).
2. In your Visual Basic code, set an ODBC connect string that references the data source. The connect string is used to establish a connection to the data source.

3 Create a new TableDef object and set its connect string to the value specified in step two.

4 Append the TableDef to the TableDefs collection.

5 Open a Recordset object on the linked table. This Recordset object contains data in your ODBC data source and you can manipulate it using the properties and methods of a Recordset object.

Each of these steps are discussed in the following sections.

Using the ODBC Data Source Manager

The first step towards accessing ODBC data is to use the ODBC data source manager to register the data source. This step is important because it stores information about the data source in the Windows Registry and makes this information available to applications.

▶ **To set up a data source**

1 In the Windows **Control Panel**, double-click on the 32-bit **ODBC** icon.

2 Click **Add...** to display the **Add Data Source** dialog, then select the ODBC driver for the data source you want to access. For example, select Microsoft SQL Server.

3 In the **Setup** dialog box, enter a data source name (DSN). This can be any string, such as "SalesDB" or "Pubs." The string doesn't have to correspond to the actual name of a database or table you want to access.

4 Enter a description of the database, such as "Sales Data for 1996." You can enter any text.

5 Enter the network name of the server on which your data source resides. Do not add a double-backslash ("\\") to the beginning of the name.

6 Enter the name of the database you'll be accessing. For example, to specify the Microsoft SQL Server sample database, enter Pubs.

Note The setup for each ODBC data source may vary because each data source driver requires a different set of information. If your data source setup dialog has values not described in the above series of steps, click the Help button on the dialog box for more information.

Registering a Data Source in Code

In some cases, you may want to register the data source in code instead of relying on users to set it up with the ODBC data source manager. To do this, use the RegisterDatabase method of the DBEngine object.

The following procedure registers a data source named PtLudlow:

```
Sub RegisterDB()
   Dim str As String

   ' Build keywords string.
   str = "Description=SQL Server on Server Puget" & _
   vbCr & "OemToAnsi=No" & _
   vbCr & "Network=(Default)" & _
   vbCr & "Address=(Default)" & _
   vbCr & "Server=Puget" & _
   vbCr & "Database=Ports"

   ' Register the database
   DBEngine.RegisterDatabase "PtLudlow", _
   "SQL Server",
      True, str
End Sub
```

Setting the Connect String

To access a data source in code, you need to set a connect string, a series of values separated by semicolons that the ODBC driver uses to locate and connect to a data source.

The connect string must start with "ODBC;", and must contain a series of values needed by the driver to access the data. The actual connect string can vary depending on the data source you're trying to access. Usually, the minimum requirement is a user ID, a password, and a DSN, as in the following example:

```
ODBC;UID=JamesK;PWD=OpenSesame;DSN=MasterData
```

When the connect string is processed by the ODBC driver and one or more of the parameters required by the data source is missing, the driver displays a dialog box asking for the information. If you don't want this dialog displayed, you must make sure that the connect string has all the needed information.

Note If you are trying to connect to a Microsoft SQL Server database that uses integrated security, set the UID and PWD values to "" (zero-length strings) because your Windows NT user name and password are automatically used. For example, the connect string might look something like the following:

```
ODBC;UID=;PWD=;DSN=MasterData
```

An example of how to use a connect string is provided in "Connection Strategies" later in this chapter.

Creating a Linked Table for an ODBC Data Source

Once you determine the proper syntax of the connect string, you can use it to create a linked TableDef object that points to data within an ODBC data source. The following code sample shows how to do this, using the Microsoft SQL Server Pubs sample database. It takes a telephone area code as its argument, links the authors table, and counts the number of authors whose phone number starts with the specified area code.

```
Public Function CountByAreaCode(strAreaCode As _
String) As Long

   Dim dbs As Database, rst As Recordset
   Dim strCn As String, tdf As TableDef
   Dim lngCount As Long

   strCn = "ODBC;DSN=Pubs;UID=SA;PWD=;DATABASE=Pubs"

   Set dbs = OpenDatabase("C:\Data\Cs.mdb")

   ' Delete any existing links named "LinkedAuthors"
   On Error Resume Next
   dbs.TableDefs.Delete "LinkedAuthors"
   dbs.TableDefs.Refresh

   On Error GoTo ErrorHandler

   ' Create a TableDef and set its Connect property.
   ' Also set the table to access:
   Set tdf = dbs.CreateTableDef("LinkedAuthors")
   tdf.Connect = strConnect
   tdf.SourceTableName = "Authors"
   dbs.TableDefs.Append tdf
   dbs.TableDefs.Refresh

   ' Open a recordset on the linked table. Use
   ' dbOpenForwardOnly for maximum speed.

   Set rst = dbs.OpenRecordset("LinkedAuthors", _
   dbOpenForwardOnly)

   Do Until rst.EOF
      If Left$(rst!phone, 3) = strAreaCode Then
         lngCount = lngCount + 1
      End If
      rst.MoveNext
   Loop
   rst.Close
   dbs.Close
   CountByAreaCode = lngCount 'Return value to caller
   Exit Function
```

```
ErrorHandler:
   MsgBox "Error " & Err & ": " & Error
   Exit Function

End Function
```

The previous example opens the Authors Recordset as a forward-only– type Recordset for maximum efficiency while moving forward without updating. To update data in the recordset, open it as a dynaset-type Recordset using the dbOpenDynaset constant in the OpenRecordset method.

A linked ODBC table behaves in a similar way to any other linked Microsoft Jet table. You can run queries against it using DAO and Microsoft Jet SQL. For example, the following code fragment creates a QueryDef that deletes records from the LinkedAuthors table:

```
Dim dbs As Database, qdf As QueryDef

Set dbs = CurrentDb
Set qdf = dbs.CreateQueryDef("DeleteAuthor")
qdf.SQL = "DELETE FROM LinkedAuthors WHERE _
au_lname = 'Smith'"
qdf.Execute
```

Using Pass-Through Queries

Although you can update information in ODBC databases using Microsoft Jet SQL update queries, or dynaset-type Recordset objects, you should consider using a SQL pass-through query, which is usually much more efficient.

SQL pass-through queries are SQL statements that the Microsoft Jet database engine sends directly to the database server without interpretation. Because they are not interpreted by Microsoft Jet, SQL pass-through queries enable your application to send server-specific commands directly to your database server.

SQL pass-through query objects are composed of two pieces of information: the SQL statement and the ODBC connection string. The SQL statement is interpreted only by the database server and must follow the server's SQL language specification. Furthermore, pass-through SQL statements cannot contain any Visual Basic functions or commands.

Advantages and Disadvantages of Pass-Through Queries

Pass-through queries offer the following advantages over other queries:

- Pass-through queries are sent to the server without modifications. More processing occurs on the server and less traffic is generated over the network.

- Pass-through queries use server-specific functionality not available from Microsoft Jet or DAO such as stored procedures, non-standard SQL extensions, server-based intrinsic functions, security, and administration commands.
- Pass-through queries log informational messages returned by the server, such as those containing error and statistical information.
- Pass-through queries are the only mechanism that allows you to join tables from different databases on the server (if your server supports multiple databases). Because each database is a separate ODBC data source, an ordinary query would consider the join to be heterogeneous and would run at least part of the query locally.
- Bulk update, delete and append pass-through action queries are faster than action queries based on linked remote tables, especially when many records are affected.

However, pass-through queries do have some disadvantages:

- A pass-through query always returns a snapshot-type Recordset, which can't be updated. An ordinary query usually returns a dynaset-type Recordset, which reflects other user's changes and can be updated.
- A pass-through action query cannot partially succeed as an ordinary action query can. In addition, you are limited to action queries that your server can run; if an action query uses multiple tables, you may have no choice but to use an ordinary query.
- With pass-through queries, you must type the SQL statement directly into the SQL Pass-Through Query Window. In contrast, you can design ordinary queries using the graphical query design grid.
- To create a pass-through query, you must know the exact SQL syntax supported by your server. SQL syntax and conventions often vary from server to server, particularly outer join syntax, intrinsic function names, wildcard characters, and the use of single versus double quotation marks. Non-pass-through Microsoft Jet queries use the same syntax, regardless of where the data is stored. Microsoft Jet automatically translates Visual Basic intrinsic function names, wildcard characters and other elements into the appropriate ODBC syntax.
- You can't call user-defined Visual Basic functions in a pass-through query, but ordinary queries can use them as if they were intrinsic Visual Basic functions.
- Pass-through queries do not automatically prompt you for parameter values like ordinary queries do.

How Pass-Through Queries Work

When your application executes a pass-through query, Microsoft Jet opens a new connection to the database server, or uses an existing connection if one is available. Then, Microsoft Jet passes the SQL statement directly to the ODBC driver, without first interpreting it as a Microsoft Jet query. The SQL statement can therefore contain server-specific commands that are unavailable in Microsoft Jet SQL.

The database server interprets the statement and returns the query results (if any) and any error or informational messages that arise during query execution. As with other queries, if there are any results to retrieve, they are retrieved on demand as required by forms or reports; this reduces network traffic by ensuring that data is only transferred when needed.

Although SQL pass-through queries bypass the Microsoft Jet query processor, they use the Microsoft Jet Recordset processor to create and manage result sets that may be generated by your query. This lets you use foreign-dialect SQL statements to take advantage of your server's query processing capabilities, while maintaining the power and ease-of-use of the Microsoft Jet Recordset model after the data has been retrieved.

Processing Multiple Result Sets

SQL pass-through queries can return more than one result set. When you assign the result set to a Recordset object, or display the results using a form or report, only the first result set of the query is processed. To remedy this situation, you must use a make-table query to capture the additional result sets into Microsoft Jet tables.

When Microsoft Jet returns the result sets from your query, it writes the results into new tables. The names of these new tables are derived from the table name you specify in the make-table query with ascending integer values following the initial table name. For example, if your query returns two result sets to the table SalesFigures, the first table would be named SalesFigures and the second would be named SalesFigures1.

Creating Pass-Through Queries With DAO

As discussed earlier in this chapter, pass-through queries have many advantages. The best way to create a pass-through query is to create a Microsoft Jet QueryDef object.

▶ To create a QueryDef object

1 Use the CreateQueryDef method to create a QueryDef object. Specify only the Name argument for the query. If you specify the SQL argument in the CreateQueryDef method, you must use only Microsoft Jet SQL syntax, which may not produce the desired results. If you want to create a temporary QueryDef, specify "" (a zero-length string) for the Name.

2 Set the Connect property of the QueryDef to a valid connect string. This tells the Microsoft Jet engine to pass the query directly to the ODBC data source without processing it.

3 Set the SQL property of the QueryDef to a valid SQL statement, using the native SQL dialect supported by the ODBC data source. For example, if you're querying Microsoft SQL Server, use Transact-SQL.

4 If your query is an action query that does not return a Recordset object, set the ReturnsRecords property of the QueryDef to False. Otherwise, set ReturnsRecords to True.

The following code sample executes three pass-though action queries against the Microsoft SQL Server Pubs sample database. The first query creates a new table called sales_archive with the same structure as the sales table. The second query inserts all sales records before 1993 into the sales_archive table. The third query deletes all sales records prior to 1993 within the sales table. Notice that a single temporary QueryDef object is used to execute all three queries.

```
Sub ArchiveSales()

   Dim dbs As DATABASE, qdf As QueryDef
   Dim strCn As String, rst As Recordset
   Dim strSQL As String

   On Error GoTo ErrorHandler

   Set dbs = CurrentDb
   strCn = "ODBC;DSN=Pubs;UID=sa;PWD=;DATABASE=Pubs"
   Set qdf = dbs.CreateQueryDef("")
   qdf.Connect = strCn

   strSQL = "CREATE TABLE dbo.sales_archive _
      (stor_id char (4) "
   strSQL = strSQL & "NOT NULL, _
      ord_num varchar (20) NOT NULL, "
   strSQL = strSQL & "ord_date datetime NOT NULL, "
   strSQL = strSQL & "qty smallint NOT NULL, "
   strSQL = strSQL & "payterms varchar (12) NOT NULL,"
   strSQL = strSQL & " title_id tid NOT NULL)"

   qdf.ReturnsRecords = False

   qdf.SQL = strSQL
   qdf.Execute

   strSQL = "INSERT INTO sales_archive SELECT * "
   strSQL = strSQL & "FROM sales WHERE "
   strSQL = strSQL & "ord_date < '1/1/93'"
   qdf.SQL = strSQL
   qdf.Execute
```

```
    strSQL = "DELETE FROM sales "
    strSQL = strSQL & "WHERE ord_date < '1/1/93'"
    qdf.SQL = strSQL
    qdf.Execute

    Exit Sub

ErrorHandler:
    MsgBox "Error " & Err & ": " & Error
    Exit Sub

End Sub
```

Note the difference between handling date data types in Microsoft Jet SQL and Transact-SQL. The last two queries in the previous example use single quotes to denote dates. In Microsoft Jet, the # character is used; these two queries would fail if they were executed against Microsoft Jet tables.

Note Always set the connect string before you set the SQL property. If you set the SQL property before setting the connect string, the Microsoft Jet database engine will try to parse the SQL string as a Microsoft Jet SQL statement.

In the previous example, the ReturnsRecords property is set to False because all three queries are action queries. In the following code fragment, a pass-through query that returns records is created. It returns all sales records in the Microsoft SQL Server Pubs sample database for the year 1994. Note that the query is executed by opening a Recordset object on the stored query definition, not by invoking the Execute method.

```
Dim dbs As DATABASE, qdf As QueryDef
Dim strConnect As String, rst As Recordset
Dim strSQL As String

Set dbs = CurrentDb
strConnect = "ODBC;DSN=Pubs;UID=SA;PWD=;DATABASE=Pubs"
Set qdf = dbs.CreateQueryDef("94Sales")
qdf.Connect = strConnect
strSQL = "SELECT * FROM sales WHERE ord_date BETWEEN "
strSQL = strSQL & "'1/1/1994' AND '12/31/1994'"
qdf.SQL = strSQL
qdf.ReturnsRecords = True

Set rst = dbs.OpenRecordset("94Sales", dbOpenSnapshot)
rst.MoveFirst
Do Until rst.EOF
    Debug.Print rst!ord_date
    rst.MoveNext
Loop
```

Note Recordset objects opened on pass-through queries are always opened as snapshots and cannot be updated.

To limit the number of records returned, set the MaxRows property of the QueryDef object. Once the query processor returns the number of records specified by MaxRows, it stops working on the query. You can also set the QueryTimeout property to limit the amount of time the query processor works on the query.

Executing Pass-Through Queries Without QueryDefs

Although creating a QueryDef object is the recommended way to execute a pass-through query, you can achieve the same result with fewer lines of code by setting the Connect property on a database object, and invoking the Execute method. The following code fragment shows how:

```
Dim dbs As DATABASE

Set dbs = OpenDatabase("c:\Data\Cs.mdb")
dbs.Connect = "ODBC;DSN=Pubs;UID=SA;PWD=;DATABASE=Pubs"
dbs.Execute "DELETE FROM TempAuthors", dbSQLPassThrough
dbs.Close
```

In this example, the database opened with the OpenDatabase method can be any valid Microsoft Jet database. The query executed has nothing to do with the database you open. The database specified in the OpenDatabase method simply provides a way to set a database variable. Once the connect string is set to the ODBC data source, the database specified in the OpenDatabase method is ignored and the query is executed against the database specified in the connect string.

You should use this method for executing pass-through queries only if the query being executed does not return records, and only if you don't need to permanently store the query in your database for later use.

Using Stored Procedures

In some environments, your access to server data is limited to a set of server-based stored procedures. All data requests and updates are carried out through these stored procedures; you have no direct access to the remote tales. In such an environment, you must use pass-through queries extensively. If you only need to view data and don't need to update it, you can create a pass-through query for each stored procedure you want to call. You can then base other queries, forms, and reports on these queries as if they were linked tables.

However, if you need to update data in this type of environment, you must collect the user's input locally and then run a pass-through query to send the data to the server, calling the appropriate stored procedure. This is typical in an environment where your server has two sets of stored procedures defined: one to retrieve data, and one to

update data. With such a scenario, you might find either of the following two approaches helpful:

- Create a form based on a local holding table in which records are temporarily stored before being sent to the server. Add code to the form's AfterUpdate and AfterInsert event procedures that call the data-updating stored procedures for each record in the holding table.
- Create an unbound form to collect data and add command buttons to the form for saving the data. You can then use the values in the form's unbound controls when you call the data-updating stored procedures.

In either case, you need to write Visual Basic code that constructs a pass-through query that uses the name of the stored procedure and the values to be supplied.

Calling Stored Procedures with DAO

Calling stored procedures using DAO is similar to creating pass-through queries. Instead of using an SQL string for the SQL property, use the name of an existing stored procedure. If the stored procedure returns records, you'll need to open a Recordset on the QueryDef that points to the stored procedure. Otherwise, you can use the Execute method for stored procedures that don't return records.

Handing Stored Procedure Return Values

In many cases, you should examine the return value from a stored procedure. This is especially important to determine the success or failure of an action query when you run it as part of the stored procedure. To handle return values, examine the first element of the Fields collection of the Recordset object on which the stored procedure was opened. For example, consider the following Microsoft SQL Server stored procedure, which increases or decreases the royalty schedules in the roysched table of the Pubs sample database. The amount of the increase or decrease is specified in the integer parameter delta:

```
CREATE PROCEDURE change_royalty @ delta int AS
UPDATE roysched SET royalty = royalty + @delta
```

The following function calls the change_royalty stored procedure shown above, examines the first element of the Fields collection, and returns 0 if execution of the stored procedure is successful.

```
Function intChangeRoyalty(intDelta As Integer) _
   As Integer
   Dim dbs As DATABASE, qdf As QueryDef
   Dim strCn As String, rst As Recordset
   Dim strSQL As String

   On Error Goto ErrorHandler
```

```
    Set dbs = CurrentDb
    strCn = "ODBC;DSN=Pubs;UID=sa;PWD=;DATABASE=Pubs"
    strSQL = "declare @status int execute @status = "
    strSQL = strSQL & "change_royalty " & intDelta
    strSQL = strSQL & " select 'return value' = "
    strSQL = strSQL & "@status"

    Set qdf = dbs.CreateQueryDef("")
    qdf.Connect = strCn

    qdf.ReturnsRecords = True

    qdf.SQL = strSQL
    Set rst = qdf.OpenRecordset(dbOpenSnapshot)

    ' Examine the first element in the recordset's
    ' Fields collection.
    ' If it is 0, the stored procedure was successful.

    If rst(0) = 0 Then
       intChangeRoyalty = 0
    Else
       ntChangeRoyalty = -1
    EndIf

    Exit Function

ErrorHandler:
      MsgBox "Error " & Err & ": " & Error
      intChangeRoyalty = -1
         Exit Function
End Function
```

Directly Accessing the Server

You can connect to a server directly in Visual Basic using the OpenDatabase method with an ODBC connection string. However, this approach has several disadvantages. Although it's possible to directly open snapshot- and dynaset-type Recordset objects on a remote database, this is much less efficient than using linked tables. Structural information about a table is stored in your database when you link that table. Having this information at a local level makes establishing a connection much faster.

Note that this speed disadvantage is realized only when you open the table that causes Microsoft Jet to establish a connection. You can avoid this by using the technique known as "preconnecting," which is described in "Connection Strategies" later in this chapter.

Connection Strategies

The first time your program issues a command that requires access to the server, a connection must be established. Establishing these connections demands time and resources from both the client and the server. This section describes several ways you can intelligently manage connections, and thus reduce the resources your application uses.

Preconnecting

Using the connection-management capabilities of the Microsoft Jet database engine, a user can log onto a server when your application starts, without waiting until the first form or datasheet opens. Microsoft Jet caches the connection and authentication information (user name and password) and reuses them as needed.

One way to preconnect is to create a form to gather user and password information, to construct a connection string containing the values, and then to use the OpenDatabase method. For example, the following procedure connects to a server called MyServer and opens a database there:

```
Sub PreConnect (strUserName As String, strPassword _
As String)
   Dim wrkRemote As Workspace
   Dim dbsRemote As Database
   Dim strCn As String

   strCn = "ODBC;DSN=Myserver;"
   strCn = strCn & "DATABASE=MyDatabase;"
   strCn = strCn & "UID=" & strUserName & "PWD="
   strCn = strCn & strPassword & ";"

   Set wrkRemote = DBEngine.Workspaces(0)
   Set dbsRemote = wrkRemote.OpenDatabase("", _
      False, False, strCn)
   ' Close the database but keep the connection.
   dbsRemote.Close
End Sub
```

If you're not sure to which ODBC data source you connect, you can display a dialog box listing all registered data sources by using "ODBC;" for the connection string argument. You can then access the actual connection string using the Connect property, which identifies your connection to the server, as in the following example. You can then use the strCn value when you create pass-through queries or link or import tables.

```
Sub PreConnect ()
   Dim wrkRemote As Workspace
   Dim dbsRemote As Database
   Dim strCn As String
```

```
    Set wrkRemote = DBEngine.Workspaces(0)
    Set dbsRemote = wrkRemote.OpenDatabase _
    ("", False, False, "ODBC;")
    strCn = dbsRemote.Connect
    ' close database but keep connection
    dbsRemote.Close
End Sub
```

In both examples, the Close method doesn't disconnect you from the server—instead Microsoft Jet caches the connection internally.

Note Because connections are shared, once you establish a connection to a server with a given user name and password, the connection will retain that identity, even if linked tables that have different stored user IDs and passwords are used. If you need varying levels of security on multiple linked tables, configure your server's security so that each individual user has the appropriate access rights, instead of designing your application around multiple identities.

Reducing Connection Use

Some servers support connections that can handle multiple partially completed queries at the same time. Other servers provide connections requiring fewer resources, but each connection can process only one query at a time (for example, Microsoft SQL Server). However, regardless of the server's capabilities, the Microsoft Jet database engine allows multiple queries and manages your connections automatically.

If your server supports multiple queries on a single connection, you generally don't have to worry about connection usage because Microsoft Jet uses just one connection to the server for the entire application. However, if your server doesn't support such queries, you should try to reduce the number of connections your application requires. Additionally, if your server has a limited number of connections due to licensing issues, you must manage connections wisely. This section describes techniques to accomplish this.

Limiting Dynasets to 100 Records

If your server uses multiple connections to process more than one query at the same time, you can reduce connection use by limiting dynaset-type Recordset objects to 100 records. A dynaset containing 100 or fewer records requires only a single connection. In contrast, a dynaset containing more than 100 records requires two connections: Microsoft Jet uses one connection to retrieve the key values from the server and another to retrieve the data associated with those keys for the records visible on the screen. Other dynasets can share the second connection, but can't share the first because all key values might not have been retrieved from the server.

As an example, three dynasets, each containing more than 100 records, require a total of four connections. When all of the key values for any of the dynasets have been retrieved, or when the dynaset is closed, the corresponding connection is released. In contrast, three dynasets, each containing 100 or fewer records, can all be handled on one connection.

Releasing Connections

Another way to reduce the number of connections is to close connections that your application is no longer using.

To cause a Recordset object to finish execution and thereby close its connections, Microsoft Jet must retrieve all records. You can cause this in one of three ways:

Place the following line in the Open event procedure of a form based on the recordset.

```
Me.RecordsetClone.MoveLast
```

- Use a TOP 100 PERCENT query (or a query's TopValues property) to return 100 percent of the records. (This is especially useful for filling list boxes and combo boxes).
- If you are using Visual Basic to manage the connection with a Recordset object, issue a MoveLast method to retrieve any remaining records.

Although they reduce connection usage, these techniques cause forms to open more slowly and are not recommended when working with large Recordset objects.

Setting the Connection Timeout

Regardless of the type of server you are using, if someone is running your application but hasn't taken any action for a long time, you may want to close connections the server held by that user. By default, connections are closed after 600 seconds (10 minutes) of idle time. You can change this value in the ...\Jet\3.5\Engines\ODBC folder of the Windows Registry by changing the ConnectionTimeout setting to the number of seconds you want to wait before closing idle connections. The three dots (...) in the path shown previously represent the location in the Windows Registry that your application uses to access the subtree Jet\3.5\Engines\ODBC.

The Jet engine closes connections even if datasheets and forms displaying remote data are still open, and then automatically reconnects when the connection is needed again.

Even if the idle time has expired, Jet can't close a connection if any of the following are true:

- A transaction is pending on the connection. You must commit or roll back the transaction before the connection can be closed.
- Results of a query are still pending. Jet doesn't close a connection if it's in the process of retrieving query results from the server.

You can cause timed-out connections to stay closed by changing the ODBC Refresh Interval setting on the Advanced tab of the Options dialog box (available through the Tools menu). This option tells the Jet engine how often to refresh a form or datasheet containing remote data. The default setting is 1500 seconds; this means that after 25 minutes of idle time, Jet refreshes the current screen of remote data from the server, which requires re-establishing a timed-out connection. If you want timed-out connections to stay closed altogether, increase the ODBC Refresh Interval setting.

Balancing Connection Use and Performance

Depending on the needs of your application, you must balance the desire for optimum performance with the need to realistically manage server resources. The number of available connections on your server is controlled by a number of factors including license counts, the number of connections in use, and the speed with which the server can service requests for connections.

You could approach connection-related performance issues by opening as many connections as your application needs at the time it is started, and keeping these connections open for the duration of that session. While this may appear to make your application run faster, it can create problems as more users request connections from the server. At some point, the server will either run out of connections, or exhaust the resources it needs to establish and manage them. In this situation, your application cannot support any more users.

You must weigh the performance advantages of keeping connections open with the capabilities of your server and the number of concurrent users using your application.

Choosing a Recordset Type

There are five types of Recordset objects: table-type Recordset objects, which are actual database tables located in the current database, dynaset-type Recordset objects, which can be updated, snapshot-type Recordset objects, which cannot be updated, dynamic-type Recordset objects, which are updatable multi-table joins, and forward-only–type Recordset objects, which cannot be updated and can be scrolled only in a forward direction.

You cannot create a table-type Recordset based on a remote data source. The decision to use any of the other four Recordset types should be based on resource and performance considerations in addition to the basic needs of your application (such as the ability to update remote data).

Dynaset-Type Recordset Objects

With a dynaset, you can extract and update data in a multi-table join, including linked tables from multiple databases. To create a dynaset-type Recordset object, use the dbOpenDynaset option of the OpenRecordset method.

When you create a dynaset-type Recordset object for records on a remote data source, only a series of bookmarks are returned. Each bookmark uniquely identifies one record in the Recordset.

The actual data in the fields of the Recordset is not returned until you specifically reference that data in your Visual Basic code. Microsoft Jet uses the bookmark to find the appropriate record and return the requested data. For best performance, only data from fields explicitly referenced in your code is returned.

For example, if the table or query represented by the Recordset object contains the fields LastName, FirstName, Phone, and Photo, the following code would only send the LastName and FirstName fields to the workstation requesting the data:

```
strLName = rstCust!LastName
strFName = rstCust!FirstName
```

The Photo field, which might contain a large bitmap, is not retrieved from the server unless it is referenced in your code.

To increase performance, you can tell a Recordset to locally cache some or all of the data it retrieves. If that data is referenced again, it is retrieved from local memory, not from the server.

For More Information See "Using Remote Data Caching in Recordset Objects" later in this chapter.

Snapshot-Type Recordset Objects

In contrast to dynasets, snapshots include a complete copy of all the requested fields in your query, whether you reference them or not. To create a snapshot-type Recordset object, use the dbOpenSnapshot option of the OpenRecordset method.

As you move through a snapshot-type Recordset for the first time, all data is copied first into memory and then, if necessary, into a temporary Microsoft Jet database in the temporary directory on the user's workstation. The resulting set of data can be scrolled forward and backward.

A snapshot derived from a Microsoft Jet data source can't be updated. However, some snapshots derived from ODBC data sources can be updated, depending on the capabilities of the back-end database.

A snapshot creates less processing overhead than the dynaset-type Recordset and is usually faster at executing queries and returning data, but it cannot be updated.

Forward-Only—Type Recordset Objects

To improve performance, you can also create a forward-only–type Recordset object, which behaves like a snapshot, but only allows you to scroll forward in a single pass through a Recordset. To create a forward-only–type Recordset object, use the dbOpenForwardOnly option of the OpenRecordset method.

When working with a forward-only–type Recordset object, you cannot use the MovePrevious or MoveFirst methods, or the Move method with a negative parameter. You also cannot use MoveLast because it implies that you have a set of records; with a forward-only–type Recordset, only one record exists at any given time.

Forward-only–type Recordset objects provide the least functionality of any Recordset object but they usually provide the greatest speed.

Dynamic-Type Recordset Objects

For maximum flexibility, you can use the dynamic-type Recordset object. Dynamic-type Recordset objects behave like dynaset-type Recordset objects but are updated dynamically as other users make modifications to the underlying tables of the Recordset. To create a dynamic-type Recordset object, use the dbOpenDynamic option of the OpenRecordset method.

For example, if you open a dynamic-type Recordset object and another user edits a record in one of the underlying tables, that change will be reflected in the Recordset you opened.

Although dynamic-type Recordset objects are the most functional type of Recordset, they may offer less performance when compared to other types of Recordset objects.

Mapping Data Types Using Microsoft Jet

Most database servers support a rich set of data types, including a number of floating point types of varying scale and precision. The set of data types supported by Microsoft Jet may not exactly match the data types on a specific database server. To enable transparent data access, Microsoft Jet must select an equivalent data type for each field in a remote table. This section discusses the rules that Microsoft Jet uses to map local data types to represent standard ODBC data types.

A Word About Floating Point Data

Techniques for handling floating point data (numeric data that does not have a fixed number of digits after the decimal point) vary among database servers. Furthermore,

when remote floating point data is coerced into a numeric format acceptable to Microsoft Jet (as in the case of linked tables), minor numeric data accuracy loss may occur. Although these accuracy losses usually occur in very large or very small floating point numbers and have no appreciable effect on common mathematical operations, they can cause a problem when used for exact comparisons. One case that is particularly important is that of floating point data in fields that are part of a unique index.

When fields containing floating point values are used as part of a unique index, and any variation (even a very small one) exists between the numeric data on the local and remote data sources, Microsoft Jet may mark the record containing the discrepancy as #Deleted. This occurs because Microsoft Jet requests the record from the server by presenting its key value. The server table doesn't contain a record with key values that exactly match the values requested by Microsoft Jet (because of the loss of accuracy), therefore it appears to Microsoft Jet that another user has deleted the record.

To remedy this situation, you can force Microsoft Jet to select another unique index (if available) that doesn't use any floating point values. Rename the new index (that is, the index that doesn't contain floating point values) so that it appears alphabetically before the original index and re-establish the link to the remote table. Microsoft Jet automatically updates the link with the new index as the primary index on the remote table.

ODBC to Microsoft Jet Data Type Mapping

For a complete description of data types recognized by ODBC, as well as a complete explanation of precision and scale in ODBC, see the *Microsoft ODBC 3.0 Programmer's Reference.*

Using ODBCDirect

ODBCDirect is implemented through existing DAO methods and properties, and as a new Connections collection of Connection objects.

A Connection object is the ODBCDirect equivalent of a Database.

Like the Databases collection, Connections is a collection of the Workspace object. A Connection object is also similar to a Database object in that it contains both the QueryDefs and Recordsets collections. This allows you to create Recordset and QueryDef objects within both the Database and Connection objects.

Note You cannot use the CreateQueryDef method and the QueryDefs collection on a Database object created from an ODBCDirect workspace. Instead, you must use then with a Connection object.

Creating an ODBCDirect Workspace

To use ODBCDirect, first create an ODBCDirect workspace with the CreateWorkspace method. The ODBCDirect workspace routes calls directly to the ODBC API, as opposed to the Microsoft Jet workspace, which first routes calls to the Microsoft Jet database engine, and then to the ODBC API. The type parameter of the CreateWorkspace method determines the type of workspace to create. To create an ODBCDirect workspace, use the constant dbUseODBC, as shown in the following code fragment:

```
Dim wks as Workspace
Set wks = DBEngine.CreateWorkspace( _
"ODBCWorkspace", "JohnSmith", "Skillet", dbUseODBC)
```

If you omit the type parameter, the default workspace type is created. The default workspace type is specified in the DefaultType property of the DBEngine object. The default value for the DefaultType property is dbUseJet, indicating a Microsoft Jet workspace. If you set the DefaultType property to dbUseODBC, all workspaces created will default to ODBCDirect if the type parameter is not specified.

In DAO, a default workspace is created as soon as it is needed. This default workspace is DBEngine.Workspaces(0), which can also be written as DBEngine(0). Setting the DefaultType property in your code before a default workspace object is needed will determine the type of the default workspace. For example, in the following code fragment, the default workspace that is created is an ODBCDirect workspace:

```
Dim wks as Workspace
DBEngine.DefaultType = dbUseODBC
DBEngine.DefaultUser = "JohnSmith"
DBEngine.DefaultPassword = "Skillet"

Set wks = DBEngine.Workspaces(0)
'wks now points to an ODBCDirect workspace
```

Newly created Workspace objects are not automatically appended to the Workspaces collection, even if you specify a name for the workspace in the name argument of the CreateWorkspace method. You can also use a zero-length string for the name argument of the CreateWorkspace method. In this case, the new Workspace object is usable, but cannot be referenced by name.

The DefaultType property can be specified without using the DBEngine object. In other words, the previous code fragment can be rewritten as follows:

```
Dim wks as Workspace
DefaultType = dbUseODBC
DBEngine.DefaultUser = "JohnSmith"
```

```
DBEngine.DefaultPassword = "Skillet"
Set wks = DBEngine.Workspaces(0)
'wks now points to an ODBCDirect workspace
```

Creating a Connection Object

Once you create an ODBCDirect workspace, the next step is to choose how to connect to your ODBC data source. With Microsoft Jet, you can make the connection through the OpenDatabase method, specifying a valid connect string that points to the data source. The OpenDatabase method returns a Database object that represents the data source.

In an ODBCDirect workspace, you have two ways of connecting to your data source: The Database object and the Connection object. These two objects are similar, but the Connection object provides the following capabilities for optimized access to ODBC data:

- Asynchronous connection — Enables your application to connect to an ODBC data source asynchronously. Instead of waiting for the connection to be established, your code can continue execution and later determine if the connection was successfully performed.
- Asynchronous query execution — Enables your application to run queries against your ODBC data source asynchronously. Instead of waiting for a long query to complete, your code can perform other tasks while the query runs.
- QueryDef objects — Enables you to define QueryDef objects that represent queries in the ODBC data source. You cannot define QueryDef objects on a Database object in an ODBCDirect workspace.

To take advantage of these capabilities, use the OpenConnection method to create a Connection object. The OpenConnection method has the following syntax and arguments:

Set *variable* = *workspace*.**OpenConnection** (*dsName*, *prompt*, *readonly*, *connect*)

The *variable* argument is the name of the new Connection object.

The *workspace* argument is the name of an ODBCDirect Workspace object from which you're creating the new Connection object.

the *dsName* argument indicates the name of the registered datasource. You can reference the new Connection object using the data source name (DSN) or the Connection object's ordinal position within its collection.

The *prompt* argument indicates whether the user can supply arguments to the ODBC connect dialogs. If you do not want the user to supply a different data source name, user name, password, or default database, use the dbDriverNoPrompt constant as the prompt argument.

The *readonly* argument controls the updatability of the data accessed through the connection. Set this argument to True to prevent updates, set it to False to allow updates.

The *connect* argument is a valid connect string that supplies parameters to the ODBC driver manager. These parameters can include user name, password, default database and data source name (DSN), which overrides the value provided in the dsName argument. Note that different ODBC data sources require different parameters in the connect argument.

The following code fragment illustrates how to use the OpenConnection method:

```
Dim cnn As Connection, strCn as String
strCn = "ODBC;dsn=Pubs;database=Pubs;uid=sa;pwd=;"
Set cnn = OpenConnection("", dbDriverNoPrompt, False, strCn)
```

Once you have created a Connection object, you can open Recordset objects and run queries on the Connection object.

Note You should use a Database object with an ODBCDirect workspace only if you want to write code that works with Microsoft Jet now, but will eventually be migrated to ODBC in the future.

Setting Default Properties

When you create a new Workspace, Connection, or Recordset object using ODBCDirect, the characteristics of the new objects are determined by the arguments in the CreateWorkspace or OpenRecordset method, or by the default properties of the DBEngine objects. The DBEngine properties that you can use with ODBCDirect are shown in the following table.

Property	Description
DefaultUser	Specifies the default user name to use when establishing a connection to an ODBC data source. The default is an empty string. This value is used only if a user name is not supplied in the *connect* argument of the OpenConnection or OpenDatabase methods.
DefaultPassword	Specifies the default password to use when establishing a connection to an ODBC data source. The default is an empty string. This value is used only if a password is not supplied in the *connect* argument of the OpenConnection or OpenDatabase methods.
LoginTimeout	Returns or sets a value that determines the number of seconds the ODBC driver waits before abandoning an attempt to connect to a data source. This value is used as an application-wide default unless the Workspace object's LoginTimeout property is used to override this value.

Property	Description
DefaultCursorDriver	Indicates whether to use a local or server cursor. The default is dbUseDefaultCursor, which lets the ODBC driver determine which type to use. In most cases, if dbUseDefaultCursor is specified, server cursors will be used if they are supported by the server. Older versions of some databases may not support server-side cursors. For example, Microsoft SQL Server 4.21 does not support cursors, but Microsoft SQL Server version 6.0 does. If the server does not support cursors, local cursors are used. To indicate a specific cursor type, set the DefaultCursorDriver property to either dbUseLocalCursor or dbUseServerCursor, but be sure that the server supports this particular cursor type.

Because you can mix Microsoft Jet and ODBCDirect workspaces in your code, you may need to determine the type of a Workspace object after it is created. This can be done using the Type property of the Workspace object. The Type property is read-only once it is set and can have either dbUseJet or dbUseODBC as its value.

Opening Connections Asynchronously

In some cases, opening connections to data sources can take a long time, forcing users to wait until the connection completes or an error occurs. To reduce the amount of time users must wait, you can open a connection asynchronously. This means that your application can complete other tasks while the connection is being established. To open a connection asynchronously, use the dbRunAsync option in combination with the prompt argument of the OpenConnection method. For example:

```
Dim wks as Workspace
Dim cnn As Connection

Set wks = DBEngine.CreateWorkspace( _
"ODBC", "sa", "", dbUseODBC)
strConnect = "ODBC;DSN=Pubs;UID=SA;PWD=;DATABASE=Pubs"
Set cnn = wks.OpenConnection("", _
dbDriverNoPrompt + dbRunAsync, _
 False, strConnect)
```

You can use the StillExecuting property of the Connection object to see if the connection has been established, or use the Cancel property of the Connection object to abort the connection attempt if it takes too long.

Running Asynchronous Queries

Creating and running queries in an ODBCDirect workspace is similar to creating and running queries in a Microsoft Jet workspace. You create the query by invoking the CreateQueryDef method on a Connection object, and then use the Execute or OpenRecordset methods on the resulting query.

You can take advantage of asynchronous queries to enable users to continue using your application while the query runs. For example, you may want to run an asynchronous query while you open a form and populate its controls. You can also give users the ability to cancel asynchronous queries if they are taking too long.

Unlike QueryDef objects created in a Microsoft Jet workspace, QueryDef objects created in an ODBCDirect workspace are always temporary—they are not saved within the data source before they are executed, even if you assign them a name.

One of the advantages of creating a QueryDef on a Connection object is that you can execute the query asynchronously by specifying the dbRunAsync constant with the Execute or OpenRecordset method, as shown in the following example:

```
Dim wks as Workspace, strSQL As String
Dim cnn As Connection, qdf As QueryDef
Dim strConnect As String, rst As Recordset

Set wks = DBEngine.CreateWorkspace( _
"ODBC", "sa", "", dbUseODBC)
strConnect = "ODBC;DSN=Pubs;UID=SA;PWD=;DATABASE=Pubs"
Set cnn = wks.OpenConnection("", dbDriverNoPrompt, _
False, strConnect)
Set qdf = cnn.CreateQueryDef("Delete94Sales")
strSQL = "DELETE * FROM sales WHERE ord_date BETWEEN "
strSQL = strSQL & "'1/1/1994' AND '12/31/1994'"
qdf.SQL = strSQL

' Run the query asynchronously
qdf.Execute dbRunAsync

' Additional code executes here while the
' query runs...
```

Note You can't use the dbRunAsync constant when executing a query using the Execute method on a Database object. Also, since you can't define a QueryDef on a Database object created from an ODBCDirect workspace, it is not possible to run asynchronous queries at all using a Database object.

In addition to using a QueryDef on a Connection object to run an asynchronous query, (as shown in the previous example), you can also use the Execute method directly on the Connection object. For example:

```
Dim cnn As Connection, strCn As String

strCn = "ODBC;DSN=Pubs;UID=SA;PWD=;DATABASE=Pubs"
Set cnn = OpenConnection("", dbDriverNoPrompt, _
False, strCn)
cnn.Execute "DELETE FROM TempAuthors", dbRunAsync
cnn.Close
```

When you run a query asynchronously, you can use the StillExecuting property to determine if the query has completed. If StillExecuting is True, the query has not yet completed; otherwise, it is False. If you want to cancel an asynchronous query, use the Cancel method. For example:

```
Dim cnn As Connection, strCn As String
DBEngine.DefaultType = dbUseODBC
strCn = "ODBC;DSN=Pubs;UID=SA;PWD=;DATABASE=Pubs"
Set cnn = wks.OpenConnection("", dbDriverNoPrompt, _
False, strCn)

' Start a transaction so we can roll back if needed.
BeginTrans
cnn.Execute "DELETE FROM TmpSales", dbRunAsync

' If the query is still running, cancel and roll back,
' otherwise, commit the transaction.
If cnn.StillExecuting Then
   cnn.Cancel
   Rollback
Else
   CommitTrans
End If

cnn.Close
```

The StillExecuting property and the Cancel method are available on QueryDef, Connection, and Recordset objects.

You can't run more than one asynchronous operation at a time on the same connection object or perform other DAO operations (such as recordset manipulation) on the same connection object while an asynchronous query runs. This is because the connection to the data source is considered "busy" while an asynchronous operation is running. To run multiple asynchronous queries on the same connection object, you must first test the value of the StillExecuting property to determine whether you can start the next asynchronous operation. To run multiple asynchronous queries in parallel, you must create separate connection objects and run each asynchronous query on its own connection object.

In most cases, you'll want to run an asynchronous query as part of a transaction. Be aware, however, that if you call CommitTrans while the asynchronous query is still running, your code will pause at the CommitTrans statement until the query finishes. For this reason, it is more efficient to periodically check the StillExecuting property and continue to perform other work while the query runs. Once StillExecuting returns False, you can then call CommitTrans. This prevents your code from pausing at the CommitTrans statement.

Note If you cancel an action query that is not part of a transaction, the query will update records up to the point where the Cancel method was called. The operation will be partially complete and will not be rolled back. For this reason, you should use the Cancel method only within the scope of a transaction. Additionally, if you start an asynchronous query in a procedure and the procedure exits before the query has completed, the query will continue to run.

In ODBCDirect queries, the CacheStart property and FillCache method are not supported as they are on Microsoft Jet queries. Instead, you should use the CacheSize method of the QueryDef object. The CacheSize method contains the number of records to cache locally. The default is 100 records. The following code fragment shows how to reset the cache size to 200 records:

```
Dim qdf As QueryDef, rst As Recordset
Dim cnn As Connection

Set cnn = OpenConnection("", dbDriverNoPrompt, False)
cnn.Connect = "ODBC;DSN=Pubs;UID=SA;PWD=;DATABASE=Pubs"
Set qdf = cnn.CreateQueryDef("tempquery")
qdf.SQL = "SELECT * FROM 94Sales"
qdf.CacheSize = 200
Set rst = qdf.OpenRecordset()
```

Opening Recordset Objects Asynchronously

In addition to running queries asynchronously, you can specify the dbRunAsync constant in the OpenRecordset method to open Recordset objects asynchronously. You can then use the Cancel method and the StillExecuting property directly on the Recordset object. For example, if you open a Recordset object asynchronously, and it takes a long time to open because more records are returned than expected, you can give users the option of canceling the operation in order to specify more restrictive criteria that returns fewer records.

If you cancel an OpenRecordset method, the Recordset object will become invalid and you must reopen it to retrieve a valid Recordset object.

Switching Between Connection and Database Objects

With ODBCDirect, you can open a Database object and a Connection object against the same ODBC data source, and use both in your code. You can then take advantage of each object for its different capabilities. You should use a Database object with an ODBCDirect workspace only if you want to write code that works with Microsoft Jet now, but will eventually be migrated to ODBC in the future.

Alternatively, you may want to create a single object and then switch to the other type when needed. To do this, use the Connection property of the Database object or the Database property of the Connection object. These properties enable you to create Connection objects from Database objects and to create Database objects from Connection objects. This is especially useful for adding ODBCDirect capabilities to existing applications that only use Database objects.

For example, you can use a Database object for most of your ODBC data access needs, but when you need to run an asynchronous query, you can create a Connection object from the Database object and then execute the query on the Connection object. The following code sample illustrates this technique.

```
Sub DeleteRecords()
   Dim dbs As Database, strCn As String
   Dim cnn As Connection

   ' Open a database in the default workspace:
   strCn = "ODBC;dsn=Pubs;database=Pubs;uid=sa;pwd=;"
   Set dbs = OpenDatabase("", False, False, strCn)

   ' Try to create a Connection object from
   ' the Database object. If we're not in an
   ' ODBCDirect workspace, an error will occur and
   ' we must execute synchronously. Otherwise,
   ' execute asynchronously:

   Err = 0
   On Error Resume Next
   Set cnn = dbs.Connection
   If Err = 0 Then
      cnn.Execute "DELETE FROM Authors", dbRunAsync
      cnn.Close
   Else
      dbs.Execute "DELETE FROM Authors"
      dbs.Close
   End If

End Sub
```

Batch Optimistic Updating

In many client/server applications, optimistic updates occur on a row-by-row basis. This usually happens with the following series of events:

1. A user edits a record.
2. The user tries to save the record.
3. The server attempts to place a lock on that record, and if successful, the record is updated. Otherwise, a lock violation is handled by the application.
4. The user moves to another record, and the entire process is repeated.

Although this process works well for many applications, it is often more efficient to have the user edit multiple records that are cached locally and then submit these records to the server in a single batch for updating. This process is called batch optimistic updating.

▶ To use batch optimistic updating

1 Create an ODBCDirect workspace.

2 Set the DefaultCursorDriver property of the workspace to dbUseClientBatchCursor. This cursor driver must be used to perform batch updates.

3 Create a Connection or Database object from the ODBCDirect workspace.

4 Use the OpenRecordset method on the Connection or Database object to open a Recordset and specify the dbOptimisticBatch constant in the *locktype* parameter.

5 Perform any edits to the Recordset object. All edits are cached locally.

6 When you are ready to update the data source, call the Update method on the Recordset object, specifying dbUpdateBatch for the *updatetype* parameter.

Note If you attempt a batch update while a record in that Recordset object is being edited by the user, the record being edited will automatically be updated before the batch update begins.

The following code sample illustrates how to use batch optimistic updating:

```
Dim wks As Workspace, cnn As Connection
Dim strCn As String

DefaultType = dbUseODBC
Set wks = DBEngine(0)
wks.DefaultCursorDriver = dbUseClientBatchCursor
strCn = "ODBC;dsn=Pubs;database=Pubs;uid=sa;pwd=;"

Set cnn = wks.OpenConnection("", _
dbDriverNoPrompt, False, strCn)
```

```
Set rst = cnn.OpenRecordset("SELECT * FROM roysched", _
dbOpenDynaset, 0, dbOptimisticBatch)

' The following loop changes all records in the
' local recordset, not the data source.

While Not rst.EOF
   rst.Edit
   rst!royalty = rst!royalty + 5
   rst.Update
   rst.MoveNext
Wend

' Now update all records in the database
rst.Update dbUpdateBatch
```

Note If multiple records have been edited locally, but you only want to update the current record before performing the batch update, you can call the Update method and specify the dbUpdateCurrentRecord constant for the *updatetype* parameter. This writes the current record to the data source without writing any other batched updates. This is illustrated in the following example:

```
Dim wks As Workspace, cnn As Connection
Dim strCn As String

DefaultType = dbUseODBC
Set wks = DBEngine(0)
wks.DefaultCursorDriver = dbUseClientBatchCursor
strCn = "ODBC;dsn=Pubs;database=Pubs;uid=sa;pwd=;"

Set cnn = wks.OpenConnection("", _
dbDriverNoPrompt, False, strCn)

Set rst = cnn.OpenRecordset("SELECT * FROM roysched", _
dbOpenDynaset, 0, dbOptimisticBatch)

' The following loop changes all records in the
' local recordset, not data source.

While Not rst.EOF
   rst.Edit
   rst!royalty = rst!royalty + 5
   rst.Update
   rst.MoveNext
Wend

' Go back to the first record, edit it, and update.
' This writes only the first record back to the
' data source.
```

```
rst.MoveFirst
rst.Edit
rst!royalty = rst!royalty + 2
rst.Update dbUpdateCurrentRecord

' Now update all records in the data source.
rst.Update dbUpdateBatch
```

Handling Collisions

When you attempt to update a group of records in a single batch operation, it is possible that other users are editing one or more records you are trying to update. If this is the case, a "collision" might occur, which prevents some of your updates from being saved in the database.

To handle collisions, examine the BatchCollisions property on the Recordset object. The BatchCollisions property is an array that stores bookmarks pointing to records in the Recordset object on which a collision occurred. Each time a collision occurs during a batch update, a bookmark for the record is added to the BatchCollisions array. You can then move to each of these bookmarks and examine the following properties on the Field object of the current record:

Property	Description
Value	The current value of the field in your Recordset object. This corresponds to the value of the field after the Update method was called.
OriginalValue	The value of the field in your Recordset object before the Update method was called.
VisibleValue	The value of the field as it is stored in the database.

After examining these properties, you can choose one of the following three options:

- Ignore the current value in your Recordset object and keep the value as it is stored in the database. To do this, set the RecordStatus property on the current record to dbRecordUnmodified, then requery the Recordset object to set the field back to its original value.
- Force the current value in your Recordset object into the database, which ignores the field's original value. To do this, call the Update method and specify True for the *force* argument.
- Change the current value in your Recordset object to the original value and force the change into the database.

Note If you call the Update method and specify dbUpdateBatch for the *updatetype* argument and True for the *force* argument, all your changes will be forced into the data source and will overwrite any changes that other users made to the records. For this reason, it is safer to call Update without specifying the *force* argument, and then resolve collisions individually using the BatchCollisions array along with the Value, OriginalValue, and VisibleValue properties.

The following code fragment shows how to use the BatchCollisions array to force all changes made to a local Recordset object into the database.

```
Set rst = cnn.OpenRecordset("SELECT * FROM roysched", _
dbOpenDynaset, 0, dbOptimisticBatch)

' Change all records in the local recordset.

While Not rst.EOF
   rst.Edit
   rst!royalty = rst!royalty + 5
   rst.Update
   rst.MoveNext
Wend

rst.Update dbUpdateBatch

' Check for collisions and force all changes
' to the recordset into the database one record
' at a time.

For j = 0 to Ubound(rst.BatchCollisions) - 1
   rst.BookMark = rst.BatchCollisions(j)
   rst.Update dbUpdateCurrentRecord, True
Next j
```

In the previous example, modifications to the Recordset object were written back to the database one record at a time. In the following example, the same result is achieved, but instead of writing one record at a time, all records are saved in a batch:

```
Set rst = cnn.OpenRecordset("SELECT * FROM roysched", _
   dbOpenSnapshot, 0 dbOptimisticBatch)

' Change all records in the local recordset

While Not rst.EOF
   rst.Edit
   rst!royalty = rst!royalty + 5
   rst.Update
   rst.MoveNext
Wend

rst.Update dbUpdateBatch, True
```

The following code fragment show how to discard changes to a Recordset object so that the database is not updated.

```
Set rst = cnn.OpenRecordset("SELECT * FROM roysched", _
dbOpenSnapshot, 0 dbOptimisticBatch)

' Change all records in the local recordset.

While Not rst.EOF
   rst.Edit
   rst!royalty = rst!royalty + 5
   rst.Update
   rst.MoveNext
Wend

rst.Update dbUpdateBatch

' Check for collisions and don't write changes
' into the database.

For j = 0 to Ubound(rst.BatchCollisions) - 1
   rst.BookMark = rst.BatchCollisions(j)
   rst.RecordStatus = dbRecordUnmodified
Next j

' Resync the recordset with the database.
rst.Requery
```

Data Pipelining

In addition to its many advantages for manipulating data within a single database, batch optimistic updating offers another advantage: the ability to move data from one database to another database with a similar schema. This capability is called *data pipelining*.

Data pipelining can simplify operations such as archiving or backing up records from one database to another.

▶ To perform data pipelining

1. Create an ODBC Workspace object and set its DefaultCursorDriver property to dbUseClientBatchCursor.
2. Open a Connection object on the Workspace object created in step 1.
3. Open a Recordset object on the Connection object created in step 2, using the dbOptimisticBatch constant for the *locktype* parameter.

4 Set the Recordset object's Connection property to a different Connection object that points to a different data source. This data source must have the same schema (table definitions) as the original data source.

5 Loop through the Recordset object and change the RecordStatus property of each record to dbRecordNew.

6 Perform a batch update by invoking the Update method on the recordset object and specifying the dbUpdateBatch constant for the updatetype parameter.

These steps are illustrated in the following code:

```
Dim cnn1, cnn2 As Connection
Dim rst As Recordset, wks As Workspace
Dim strCn1 As String, strCn2 As String

DefaultType = dbUseODBC
Set wks = DBEngine(0)
wks.DefaultCursorDriver = dbUseClientBatchCursor

strCn1 = "ODBC;dsn=Pubs;database=Pubs;uid=sa;pwd=;"
strCn2 = "ODBC;dsn=Pubs95;database=Pubs;uid=sa;pwd=;"

Set cnn1 = wks.OpenConnection("Conn1", _
dbDriverNoPrompt, False, strCn1)

Set cnn2 = wks.OpenConnection("Conn2", _
dbDriverNoPrompt, False, strCn2)

strSQL = "SELECT * FROM sales WHERE ord_date BETWEEN "
strSQL = strSQL & "'1/1/1995' AND '12/31/1995'"

' Open a recordset containing 1995 sales data.

Set rst = cnn1.OpenRecordset(strSQL ,dbOpenDynaset, _
0, dbOptimisticBatch)

' Bind the recordset to a different data source.
Set rst.Connection = cnn2

' Change each row's status to a new row, indicating
' that a batch update should insert new rows for all
' rows in the recordset.

While (rst.EOF <> True)
   rst.RecordStatus = dbRecordNew
   rst.MoveNext
Wend

' Perform a batch update. This copies all records
' to the data source pointed to by cnn2.

rst.Update dbUpdateBatch
```

Cursors

A cursor is a pointer or set of pointers to data extracted from a database that meets specified criteria. Most modern database engines use cursors as a way to point to returned data. Microsoft Jet employs a cursor model that maintains an ordered set of pointers to actual data. Microsoft Jet cursors are available to you through the Recordset object.

Characteristics of Cursors

Cursors allow you to work with sets of data. Cursors are generally a representation of real data, and can have any of the following attributes:

- They can represent some or all records in a single table.
- They can represent some or all records in a multi-table join.
- They can represent no records.
- They can be read-only or updatable at either the cursor or the field level.
- They can be fully scrollable (meaning that you can move forwards and backwards through the records), or they can be forward-only scrolling.
- They can exist on either the client, or the server.
- They can use actual data, or keysets.

Cursor Models

A cursor requires temporary resources to hold its data. These resources can be in the form of RAM, a paging file (such as the virtual memory feature of Microsoft Windows), or temporary files or databases. If these resources are stored on the client machine, the cursor is known as a *client-side* cursor. With this type of cursor, the server sends the data that the cursor represents across the network to the client, along with the data required by the cursor itself. It is the responsibility of the client to manage the temporary resources needed by the cursor.

Microsoft SQL Server version 6.0 (and some other server database engines) support an additional type of cursor known as *server-side* cursors. With this cursor type, the server is responsible for managing temporary objects using resources located on the server itself. With server-side cursors, only cursor-selected record data is returned to the client over the network. As a result, the use of this type of cursor can result in significant performance improvements compared to client-side cursors, especially in situations where excessive network traffic or inadequate network bandwidth are problems. However, since resources in the form of RAM and disk space are needed at the server, you must plan accordingly and ensure that your server hardware is capable of managing all cursors requested by clients.

Choosing a Cursor Type

When you open a Recordset object, you can specify different constants that determine the type of cursor represented by the Recordset object. The following table shows the four constants you can use for the *recordsettype* parameter and the type of cursor each of these represents:

Constant	Cursor Type
dbOpenDynamic	Dynamic
dbOpenDynaset	Keyset
dbOpenSnapshot	Static
dbOpenForwardOnly	Forward-only scrolling. This is the default.

For detailed information about ODBC cursors, see the *Microsoft ODBC 3.0 Programmer's Reference.*

With the DefaultCursorDriver property of the Workspace object, you can choose where the cursor keyset will be created — on the workstation or the server. The DefaultCursorDriver property can be assigned any of the following values:

Constant	Description
dbUseLocalCursor	Specifies that local cursors should be used. This option gives better performance for small result sets, but degrades quickly for larger result sets.
dbUseServerCursor	Specifies that server-side cursors should always be used. For most large operations this usually offers better performance.
dbUseDefaultCursor	Let the ODBC driver determine which driver to use.
dbUseClientBatchCursor	Specifies that the client batch cursor library should be used. This option is required for batch updates.
dbUseNoCursor	Opens all cursors (that is, Recordset objects) as forward-only type, read-only, with a rowset size of 1. Also known as "cursorless queries."

If your data source supports server-side cursors, the remote database engine builds keysets on the server and extends the SQL dialect to support creation and management of data using cursors. Microsoft SQL Server 6.0 supports server-side cursors, but not all data sources do. Use the DefaultCursorDriver property of the DBEngine object to choose the type of cursor driver to use.

Server-side cursors reduce the amount of memory and disk space required on the client machines, but shift that load to the server. For Microsoft SQL Server 6.0, cursor keysets are maintained in the TEMPDB database on the server, so it must be sized to meet the needs of additional temporary cursor keysets generated by multiple clients.

Record Locking

When you open a Recordset object, you can also specify the type of record locking you want to use. This is done by setting the *locktype* parameter to the appropriate constant value. The following table show the four types of locking supported by Recordset objects, and the ODBC cursor lock type to which they correspond:

Constant	ODBC Locking Type
dbOptimistic	Corresponds to optimistic concurrency using row versions (**concur row version**). The ODBC cursor library and the data source compare record D or TIMESTAMP values to determine if the record has changed.
dbPessimistic	Corresponds to pessimistic locking (**concur lock**). This option locks the row or the data page containing the row(s) as soon as the Edit or AddNew method is executed, and holds it until the Update method has written the changes to the data source.
dbOptimisticValue	Corresponds to optimistic concurrency using row values (concur values). The ODBC cursor library and the data source compare data values.
dbReadOnly	No updates are permitted (read only). This is the default.

When values or record versions are compared to test concurrency, the original row as it was prior to the Edit or AddNew method is compared with the row as it exists in the database prior to being changed by the Update method. If the row data changed in this time frame, a trappable error results. At this point, your temporary row buffer is lost. To carry out the changes, you must refresh the current row using the Move 0 technique, use the Edit or AddNew method, fill in the new values, and repeat the Update method again.

Some combinations of cursors and lock types will not work together. For example, with Microsoft SQL Server 6.0 cursors, dbOpenSnapshot only supports dbReadOnly. Precisely which combinations work together depends on the cursor driver. Refer to your cursor driver documentation for specific information about compatible lock types.

Your cursor driver can handle different combinations of cursor types and lock types in different ways. In some cases, it may return an error if it does not handle a specific combination. In other cases, it may switch to the nearest possible combination that it supports. If an error occurs, DAO will place the error information in the Errors collection.

Note The dbOpenTable constant is not supported on ODBCDirect Recordset objects and the dbOpenDynamic constant is not supported on Microsoft Jet Recordset objects.

Cursor Limitations

If you open a Recordset in a Microsoft Jet workspace without specifying any of the constants in the OpenRecordset method, DAO defaults to a dynaset-type Recordset with optimistic locking, which provides the greatest functionality and programming flexibility.

In an ODBCDirect workspace, the default is not the most functional Recordset, but the fastest, which is a read-only, forward-scrolling-type Recordset. Therefore, if you want to edit data on the server, you'll need to explicitly specify a lock type other than dbReadOnly for the *locktype* parameter of the OpenRecordset method.

ODBCDirect Recordset objects do not support the Index property or the Seek method. This is because tables in remote ODBC data sources are never opened directly, which is required when performing indexed seeks.

Recordsets opened against ODBC data sources also do not support any of the Find methods: FindFirst, FindNext, FindPrevious, FindLast.

Handling Multiple Result Sets

Any SQL statement can include multiple SELECT statements or stored procedures that invoke one or more SELECT statements. Each SELECT statement generates a result set that must be processed by your code or discarded before the resources are released and the next result set is made available.

Action queries also generate row-less result sets that must also be processed. For example, if you submit a query that includes four SELECT queries to populate four local ListBox controls, and a stored procedure that updates a table, your code must deal with at least five result sets. Because you don't necessarily know how many results sets will be generated by a stored procedure, your code must be prepared to process an unknown number of result sets.

You can use either local cursors or server cursors to retrieve multiple result sets. If you use local cursors, multiple result sets will be returned no matter what type of Recordset object you request. If you use a server-side cursor to retrieve multiple result sets, you should follow this procedure.

▶ To retrieve multiple result sets with a server-side cursor

1. Set the workspace's DefaultCursorDriver property to dbUseLocalCursor.
2. Create a QueryDef object and set its SQL property to a valid SQL string that returns multiple Recordset objects.
3. Set the CacheSize property of the QueryDef to 1. This tells the server to send you one record at a time (cursors are not used).

4 Open a Recordset object on the QueryDef you just created, specifying dbOpenForwardOnly and dbReadOnly for the *recordsettype* and *locktype* parameters.

5 Use the NextRecordset method to access the next Recordset object in the group of Recordset objects returned from the server. This discards the current Recordset object and replaces it with the next Recordset object specified in your query's SQL statement. If there are no more Recordset objects in the group of Recordset objects, the return value of the NextRecordset method will be False and the current Recordset object will be empty.

Note An important consideration when handling multiple result sets is that none of the result sets can be updated.

```
Sub GetMultipleResults()
   Dim rst As Recordset, cnn As Connection
   Dim strCn As String
   DefaultType = dbUseODBC
   DBEngine(0).DefaultCursorDriver = dbUseLocalCursor
   strCn = "ODBC;dsn=Pubs;database=Pubs;uid=sa;pwd=;"
   Set cnn = OpenConnection("", dbDriverNoPrompt, _
   False, strCn)
   strSQL = "SELECT * FROM Authors; "
   strSQL = strSQL & "SELECT * FROM Titles;"
   Set rst = cnn.OpenRecordset(strSQL)

   Do While rst.NextRecordset() = False
      While Not rst.EOF
         For Each fld in rst.Fields
            Debug.Print fld.Name, fld.Value
         Next fld
         rst.MoveNext
      Wend
   Loop
   cnn.Close

End Sub
```

Working with Stored Procedures

You can use ODBCDirect QueryDef objects to execute stored procedures. ODBCDirect QueryDefs objects support stored procedures that have input and output parameters, as well as return values.

The following code fragment shows how to create a stored procedure on the server:

```
strSQL = "CREATE PROC GetEmps AS "
strSQL = strSQL & "SELECT * FROM EMPLOYEES;"
cnn.Execute strSQL
Set qdf = cnn.CreateQueryDef("qry", _
"{call GetEmps()}")
Set rst = qdf.OpenRecordset
```

Use the Parameters object to work with parameters. The Direction property tells DAO how the parameter will function. The ODBC driver will try to determine the parameter direction, but the property is writable so you can set it if needed. The following example creates a simple stored procedure with an input parameter and a return value. It then executes the procedure and retrieves the return value:

```
strSQL = "CREATE PROC UpdateEmps (@invar int) _
" AS RETURN @invar;"
cnn.Execute strSQL

'Create a QueryDef to execute the stored proc.
Set qdf = cnn.CreateQueryDef("qry", _
"{? = call UpdateEmps(?)}")

'Handle the parameters.
qdf.Parameters(0).Direction = _
dbParamReturnValue
qdf.Parameters(1) = 10
qdf.Execute

' Get the return value.
var = qdf.Parameters(0).Value
```

Optimizing Performance

In addition to functional differences between local and remote tables, there are often considerable performance differences. An incorrectly designed and implemented client/server application can cause the application to fail to meet its goals. This lack of understanding of the basic concepts of client/server design can eventually lead to the incorrect dismissal of client/server as a viable architecture.

Because of this, you must pay close attention to how you design and develop your client/server application. You want to have a good understanding of the differences between the file-server and client/server architectures, and a good understanding of the common performance optimization techniques available. This section discusses techniques you can use to improve the performance of your client/server applications.

Speeding Up Queries

The best way to improve the performance of queries on remote data is to have the server execute as much of the query as possible. Microsoft Jet attempts to send the entire query to the server, but evaluates locally any query clauses and expressions that aren't supported by servers in general or by your server in particular. Some information about a server's capabilities are available when you link a remote table.

If a query contains expressions that can't be evaluated by your server, more or less of the query is evaluated locally, depending on where the expressions occur. Queries with such expressions in the SELECT clause are still evaluated on the server, unless they occur in a totals query, a DISTINCT query, or a UNION query, in which case they are evaluated locally. Such expressions in other clauses (for example, WHERE, ORDER BY, and GROUP BY clauses) cause at least part of the query to be evaluated locally.

This section describes how to design queries that maximize the use of the server and thereby improve query performance.

Using Supported Functionality

If possible, don't design queries that use functionality not generally supported by servers. The following sections identify some areas of functionality that aren't generally supported and others that usually are.

Functionality Not Supported by Most Servers

Most servers don't support the following:

- Operations that can't be expressed in a single SQL statement — for example, when a query's FROM clause contains a totals query or a DISTINCT query. Often, you can rearrange your queries to calculate totals or apply the DISTINCT predicate after all other operations.
- Operations that involve Microsoft Jet-specific extensions to SQL, such as complex crosstab queries, TOP queries (queries using the TopValues property), and reports with multiple levels of grouping or totals
- Expressions that contain operators or functions specific to Microsoft products. For example, the DAO financial and statistical aggregate functions have no server equivalents.
- User-defined functions in Visual Basic that take remote fields as arguments. These functions don't exist on the server, but must process remote field data.
- Expressions or UNION queries that mix text and numeric data types. Most servers lack the data type-matching flexibility of Microsoft Jet. If you need to mix data types, use explicit conversion functions.
- Heterogeneous joins between local tables and remote tables, or between remote tables in different ODBC data sources.

Functionality Supported by Some Servers

Servers differ in certain areas of functionality. Some servers support the following capabilities:

- Outer joins (Note that Microsoft Jet doesn't send multiple outer joins to a server, although many inner joins can accompany a single outer join.)
- Numeric, string, and date/time functions (such as Log, Mid$, and DatePart).
- Conversion functions (such as CInt, CStr, and CVDate).

If the operation is supported, it is sent to the server for processing; otherwise, it's performed locally.

Splitting WHERE Clause Restrictions

If a query's SQL statement includes a WHERE clause that the server can't evaluate completely, Microsoft Jet splits the clause into its conjuncts (linked by AND), and sends as many conjuncts to the server as possible. For example, suppose you've written a function in Visual Basic called MyFunction and run the following query:

```
SELECT field1, field2, field3
   FROM table1
   WHERE (field1 = 100 OR field1 = 200)
   AND MyFunction (field2) > 100
   AND field3 LIKE "A*"
```

Microsoft Jet sends the following query to the server:

```
SELECT field1, field2, field3
   FROM table1
   WHERE (field1 = 100 OR field1 = 200)
   AND field3 LIKE "A*"
```

Microsoft Jet evaluates `MyFunction(field2) > 100` locally on the records returned.

If you need to use a WHERE clause restriction that your server can't process, you can limit the amount of data that Microsoft Jet requests by providing additional restrictions that your server can process. For maximum efficiency, these restrictions should involve fields indexed on the server.

Optimizing Expressions Containing Elements Specific to Microsoft Jet

If expressions in your queries contain user-defined functions, domain aggregate functions, or other elements specific to Microsoft Jet (such as IIf and Choose), the server cannot run them completely. If an expression using any of these elements involves remote data fields, the Jet engine runs it locally and retrieves all necessary data from the server. If such an expression involves only constants and query parameters, however, Jet evaluates it just once and sends the result to the server as a query parameter.

Whenever possible, structure your queries to take advantage of this optimization by avoiding references to remote fields in expressions that the server can't evaluate. As an example, suppose you have a remote Tasks table with a Status field containing code numbers (1 means unfinished and 2 means complete). The following query requires a user to type 1 or 2, which isn't very intuitive:

```
SELECT * FROM Tasks
   WHERE Status = [What Status?]
```

In contrast, the following query allows the user to type **Unfinished** or **Complete** when prompted:

```
SELECT * FROM Tasks
   Where [What Status?] = IIf(Status=1,'Unfinished','Complete')
```

However, since the IIf function uses the remote Status field, the Jet engine sends the first part if the statement (`SELECT * FROM Tasks`) to the server and evaluates the restriction on each returned record locally.

The following query also allows the user to type words instead of numbers:

```
SELECT * FROM Tasks
   WHERE Status = IIf([What Status?]='Unfinished',1,2)
```

However, because the return value of the IIf function is effectively constant, Jet evaluates it once locally, sends the following statement to the server, and supplies this part of the expressions as a parameter value:

```
SELECT * FROM Tasks
   WHERE Status = ?
```

Obviously, this method is more efficient and reduces network traffic.

Heterogeneous Joins

To perform a join between local and remote tales, or a *heterogeneous join*, Microsoft Jet does one of the following:

- Requests all the records in the remote table or tables and performs the join locally.
- Performs a *remote index join.* This means that for each key in the local table, Microsoft Jet asks the server only for the records with a matching key value.

Microsoft Jet uses a remote index join only if the remote field being joined is indexed and if the local table is considerably smaller than the remote table. For example, to join a local table called Employees containing 10 records and a remote table called

Tasks containing 50 records on the EmployeeID field, Microsoft Jet retrieves the 50 records from the Tasks table on the server and processes the join locally. However, if the remote table contains 1000 records, Microsoft Jet sends 10 of the following queries to the server, supplying each EmployeeID value from the local Employees table as a parameter value:

```
SELECT EmployeeID, TaskID
   FROM Tasks
   WHERE EmployeeID = ?
```

This is generally much faster than retrieving all 1000 records.

Always index the remote join field to enable Microsoft Jet to perform a remote index join when appropriate. Regardless of whether or not Microsoft Jet can perform a remote index join, you can improve the performance of a query that uses a heterogeneous join by supplying additional restrictions on remote fields. Microsoft Jet sends these restrictions to the server, limiting the amount of data requested for the join.

Speeding Up Forms

When you design a form based primarily on server data, it is usually wise to determine what data and functionality is needed, and design forms that delay asking the server for this data and functionality until it is explicitly requested by the user. This section presents several techniques that illustrate this technique.

Requesting Less Data

The more data your application requests from the server, the more network traffic you generate and the more of a load you place on the server. To minimize these unwanted effects, consider the following:

- Request as few records as necessary. For example, rather than opening a large recordset and navigating with one of the Find methods, use a filter or query to limit the number of records in the recordset, taking care to use restrictions that the server can process remotely.
- Base your forms on queries that select as few remote fields as necessary. That way, only the fields used on the form are actually retrieved from the server. If you need to add fields to the form later, you can use the Query Builder (available from the form's RecordSource property box).
- User fewer bound controls, such as text boxes, list boxes and combo boxes. Each form control that is bound to a remote field requires a separate query to be sent to the server when the form is opened. In particular, avoid controls containing totals, and list boxes and combo boxes whose RowSource properties request a large number of records.

Displaying Fields Only On Request

In some cases, you may want to display all a field's records on a form. In other cases, you might not need to make all the fields available to the user. For example, because Memo and Long Binary (OLE Object) fields can contain a large amount of data, displaying them on a form only when the user explicitly requests them can improve performance. In this example, consider removing the Memo or Long Binary (OLE Object) field from your form and placing it in a dialog form that is only opened when the user presses one of the form's command buttons.

To speed up processing of your forms, you can use the following techniques to avoid retrieving unnecessary data:

- Set the Visible property to No for controls bound to Memo and Long Binary fields, and add a toggle button or command button that sets the property to Yes when the user wants to view the contents of these controls.
- Display the most important fields on a main form and provide a button labeled "More Information" that opens another form containing other fields. Base the second form on a query that is parameterized by a unique key field on the main form.

Downloading Data For Use in More Than One Form

Often, an application contains several forms that use the same remote table — for example, as the source for a list box or combo box. If the data in the table doesn't change frequently, you can speed up form loading and reduce both network traffic and server load by considering the following techniques:

- If the table contains a small amount of data that never changes (such as the names and two-letter abbreviations of the 50 states), you should store it in the local Microsoft Jet application database. If the table is joined in queries with remote tables, you should also keep a copy of it on the server to avoid heterogeneous joins.
- If the data in the table changes rarely (such as a list of the buildings that a company owns), it should reside both on the server and in the local application database. Your application should provide a way for the user to download the current data when it changes.
- If the data in the table changes occasionally (that is, less than daily, such as the employees in a small company or department), you should store it both on the server and in the local database. In addition, your application should provide an automated mechanism that detects when the two copies are not synchronized and automatically downloads the current data from the server. This technique uses extra time when the application starts, but offers performance boosts once the application is running.

In the following example, the Depts and Emps tables are local versions of the linked server tables RmtDepts and RmtEmps:

```
Dim wrkRemote As Workspace
Dim dbsRemote As Database

Set wrkRemote =DBEngine.Workspaces(0)
Set dbsRemote = wrkRemote.Databases(0)

wrkRemote.BeginTrans
   With dbsRemote
      .Execute "DELETE FROM Depts"
      .Execute "INSERT INTO Depts SELECT * _
       FROM RmtDepts"
      .Execute "DELETE FROM Emps"
      .Execute "INSERT IN?TO Emps SELECT * _
       FROM RmtEmps"
   End With
wrkRemote.CommitTrans
```

This example empties the Depts and Emps tables and then re-populates them with the contents of the RmtDepts and RmtEmps tables on the server.

Using Less Functionality

To improve the performance of your client/server application, you may want to avoid using some of the more powerful features of Microsoft Jet. This section discusses when you can improve performance by using less functionality in your forms.

Snapshots Versus Dynasets

If you don't need to update data with your form, and the Recordset contains fewer than 500 records, you can reduce the time it takes to open the form by setting the form's AllowUpdating property to No Tables. This causes Microsoft Jet to use a snapshot as the record source for the form which is generally faster than using a dynaset.

If you only need to make one pass through the data, you can make it even more efficient by opening a forward-only–type Recordset object. In this Recordset type, the resulting data isn't copied to a scrollable buffer, which reduces the amount of time and resources necessary to create the Recordset object.

However, if the Recordset you're using is large or contains Memo or Long Binary fields, using a dynaset is more efficient. If you use a dynaset, Microsoft Jet retrieves only the primary key of each record; if you use a snapshot or forward-only type, it retrieves the entire record. A dynaset is also more efficient because Memo and Long Binary fields are retrieved only if they are displayed on the screen, or directly referred to in Visual Basic code.

Multiple-Table Queries Versus Subforms

Two methods are available for displaying information from multiple tables in a form — you can either base the form on a query that joins the tables, or you can place one or more subforms on the form. The first approach usually sends a single query to the server, which creates the join. The second approach sends at least two queries: one to retrieve the data for the main form, and one to retrieve the data for each of the subforms, parameterized by the linking fields. This second approach requires more time for the form to load as well as more overhead on the part of the server.

Because you can update and insert data into almost every field of a multiple-table query, a subform is often necessary. If you don't need the convenient display capabilities of a subform, base your form on a single query for better performance.

Improving the Performance of Updates and Deletions

Perhaps the simplest way to improve the performance of updates and deletions — especially if your remote table has many fields — is to add a *version field* (sometimes called a *timestamp*) to the remote table. A version field is maintained by the server, and its value automatically changes each time the record is updated. Typically, you can't read the contents of the version field, but its behavior enables Microsoft Jet to detect changes in records.

When updating or deleting a record, Microsoft Jet checks to see if the version field has changed. If it has, the update or deletion is canceled to avoid overwriting another user's changes. If the table doesn't have a version field, Microsoft Jet compares all the old field values to their current values to determine if data has changed. This is less efficient and generates more network traffic. Moreover, when Microsoft Jet checks values, data that hasn't changed occasionally appears as if it had, either because of inaccuracies inherent in comparing floating-point numbers, or because of the fact that Memo and Long Binary fields aren't compared at all.

To add a version field to a remote table, you can use the ALTER TABLE statement. For example, the following command adds a version field to a SQL Server table:

```
ALTER TABLE RemoteTable ADD VersionCol TIMESTAMP
```

You can run this statement as a pass-through query in Microsoft Jet, or use the administration tools available on your server for adding fields. Note that you must re-link the remote table to inform Microsoft Jet of the new field's existence.

Note Not all servers support version fields. To see if yours does, check the server's documentation.

Using Remote Data Caching in Recordset Objects

Another way to improve your application's performance is to cache remote data. Datasheets and forms based on remote data automatically cache data, but dynasets created in Visual Basic do not. If you are using Visual Basic code, and you'll be moving around in and retrieving data from a small range of records (less than 200), you should explicitly use remote data caching. You can use the CacheStart and CacheSize properties to specify the range you want to cache within the Recordset. Use the FillCache method to quickly fill all or part of this range with data from the server.

Caching also helps when you simulate an action query by using a Recordset. You might do this if the operations involved are too complex for an action query. Typically, to simulate an action query, you advance through each record in the Recordset, analyze the record's contents, and possibly update or delete the record. You can greatly improve performance if you move and fill the cache before processing every 100 records, even if you make only one pass through the data.

Using the MSysConfig Table to Control Background Population

You can control the rate at which Microsoft Jet reads records from the server during idle time by creating a table named MSysConf on your server. This table enables you to set the number of rows of data that are retrieved at one time and the number of seconds of delay between each retrieval. If you're experiencing excessive read locking on the server, you can adjust the table settings to increase background population speed. If too much network traffic is generated by background population, adjust the settings to slow it down.

Recordset objects you create using Visual Basic aren't populated during idle time. If your server places read locks on records as they are retrieved, you shouldn't remain on a record or page for an extended time. Partially populated Recordset objects in you Visual Basic code can prevent other users from updating. If the Recordset is small, you can use the MoveLast method to fully populate the Recordset and release any such locks. If the Recordset is large, you may want to use a Timer event procedure on a form to implement you own version of background population.

Other Issues in Client/Server Design

Because of the architectural differences between file-server databases and client/server databases, there are several areas of application development you should be familiar with to implement a successful client/server application. This section covers areas not considered in the previous topics.

Using Transactions

A powerful way to improve the robustness and multi-user concurrency of your client/server applications is to use transactions. Transactions enable the Microsoft Jet database engine to accumulate multiple updates and write them as a single batch. By performing operations as a batch, transactions also ensure that operations either succeed or fail as a group, and they shield other users from partially completed updates.

You can control transactions in Visual Basic by using the BeginTrans, CommitTrans, and Rollback methods of the Workspace object. Consider using transactions in the following areas:

- A sequence of Recordset updates, insertions, or deletions, or any combination of the three.
- A sequence of action queries.
- A sequence of pass-through queries that change data.
- Any combination of these operations in a sequence.

On most servers, transactions generate locks that prevent other users from updating or even reading data affected by the transaction until it's committed or rolled back. Therefore, keep your transactions as short as possible, and avoid placing code sequences that wait for a user's input inside a transaction.

How Microsoft Jet Uses ODBC Transactions

When your application does not explicitly use transactions through Visual Basic code, Microsoft Jet uses ODBC in *auto-commit* mode. In this mode, each SQL statement sent to the server has an implicit transaction around it, that is, the SQL statement has an immediate effect on the ODBC data and cannot be rolled back. Microsoft Jet stops using auto-commit mode and uses explicit ODBC transactions in the following situations:

- SQL statements that modify data (for example, UPDATE, INSERT, or APPEND).
- Explicit DAO transaction methods used through Visual Basic.

When using "bulk operation" SQL statements such as UPDATE, INSERT, and APPEND, Microsoft Jet places a transaction around the operation so that in can succeed or fail as a single entity. This is necessary because one such Microsoft Jet SQL statement may actually correspond to many individual server statements.

Note The ODBC transaction model does not support nested transactions. That is, you cannot execute a second BeginTrans method before the previous transaction is either committed or rolled back. However, if your ODBC data source supports it, you can use SQL statements to execute nested transactions.

When you use ODBC transactions, all databases in the specified Workspace object are affected — even if multiple Connection or Database objects are opened in the workspace. For example, suppose you use a BeginTrans method against one of the databases visible through the connection, and you update several rows in the database, and then delete rows in another Connection database. When you use the Rollback method, both the update and delete operations are rolled back. You can create additional Workspace objects to manage transactions independently across Connection objects. Transactions executed by multiple Workspace objects are serialized and are not atomic operations. Because of this, their success or failure is not interdependent.

Bulk Updates

The following example demonstrates how to use a transaction in Visual Basic to perform multiple updates in a group. It uses three pass-through queries in a transaction to transfer money from a savings account to a checking account. The Visual Basic application sends queries and updates to the server and retrieves the data that it needs.

```
Sub TransferFunds()
   Dim wrk As Workspace
   Dim dbsTransaction As Database
   Dim qdfUpdate As QueryDef, str As String

   On Error GoTo TransferFailed

   Set wrk = DBEngine.Workspaces(0)
   Set dbsTransaction = wrk.Databases(0)
   wrk.BeginTrans     ' Begin Transaction

   ' Create pass-through query.
   Set qdfUpdate = dbsTransaction.CreateQueryDef ("")

   str = "UPDATE Accounts SET Balance = Balance - 100"
   str = str & " WHERE AccountID = 'SMITH_SAV'"

   With qdfUpdate
      .Connect = _
      "ODBC;DSN=Bank;UID=teller;DATABASE=access"
      .ReturnsRecords = False
      .SQL = str
       .Execute' Subtract from savings account.
```

```
        str = "UPDATE Accounts SET Balance = Balance -"
        str = str & " 100 WHERE AccountID = 'SMITH_CHK'"
        .SQL = str
        .Execute     ' Add to checking account.

        str = "INSERT INTO LogBook "
        str = str & "(Type, Source, Destination,
        str = str & " Amount) VALUES ('Transfer', "
        str = str & " 'SMITH_SAV', 'SMITHC_CHK', 100)"
        .SQL = str
        .Execute       'Log transaction.
    End With
    wrk.CommitTrans' Commit transaction.
    Exit Sub

TransferFailed:
    Msgbox Err
    wrk.Rollback    ' Roll back if any Execute fails.
End Sub
```

Note the use of the CreateQueryDef method to create a temporary pass-through query. This technique introduces the minimum possible overhead for the Microsoft Jet database engine, while still making use of transactions transparently through Visual Basic. However, it limits functionality to what the server provides, and it is more difficult to define a parameter query this way.

Bulk Insertions

If your application provides data-entry forms for multiple records of data, such as an order entry system, you can improve performance and robustness by saving new records in a local table and then transferring groups of records from the holding table to the server all at once within a transaction.

▶ To insert a group of records on the server using a transaction

1. Create empty local tables containing the same fields as your remote tables.
2. Create a data-entry form bound to these local tables.
3. On the form, create a command button labeled "Post Records" for sending records to the server.
4. Create an event procedure for the button's Click event that copies the local records to the server tables and deletes the records from the local tables. The event procedure should perform all insertions and deletions within a transaction.

For example, suppose you have an Orders form containing an Order Details subform based on two local tables, LclOrders and LclOrderDetails. The user enters a new order and a group of detail records in the subform. When the user clicks the command button labeled "PostRecords", the following procedure is run:

```
Sub PostRecords_Click ()
   Dim wrkTrans As Workspace
   Dim dbsPosting As Database
   On Error GoTo TransferFailed

   Set wrkTrans = DBEngine.Workspaces(0)
   Set dbsPosting = wrkTrans.Databases(0)
   With dbsPosting
      .Execute "INSERT INTO RmtOrdersEmpty " _
      & " SELECT * " & _
      "FROM LclOrdes", dbFailOnError
      .Execute "INSERT INTO RmtOrderDetailsEmpty "_
      & SELECT * " & _
      "FROM LclOrderDetails", dbFailOnError
      .Execute "DELETE FROM LclOrders"
      .Execute "DELETE FROM LclOrderDetails"
   End With
   wrkTrans.CommitTrans ' Commit the transaction.
   Me.Requery      ' Clear form for next order entry.
   Exit Sub

TransferFailed:
   Msgbox Err
   ' Roll back if any Execute fails.
   wrkTrans.Rollback
    Exit Sub
End Sub
```

This examples uses the dbFailOnError constant with the two append queries so that an error occurs if any part of either query fails. The error handler then rolls back any changes made. Also note the use of Requery method after deleting the records from the local tables to present the user with a blank form for the next entry.

This procedure doesn't insert the records directly into the tables on the server. RmtOrdersEmpty and RmtOrderDetailsEmpty aren't the remote tables; instead, they are the following queries, which are based on linked remote tables but don't return records:

```
SELECT * FROM RmtOrders WHERE False
SELECT * FROM RmtOrderDetails WHERE False
```

For these append queries, inserting records into empty queries on linked remote tables provides the fastest possible speed, similar to opening a Recordset using the OpenRecordset method with the dbAppendOnly constant.

Limitations of Transactions

Although using transactions can improve the robustness and performance of your client/server applications, they do have several limitations:

- Although you nest transactions in Visual Basic, servers generally don't support nested transactions. The Jet engine sends only the first-level transaction to the server.
- Depending on your server, some pass-through queries may not be allowed within a transaction. For example, some servers don't allow data-definition queries within transactions. Check your server's documentation to identify these limitations.
- Avoid sending server-specific transaction commands in pass-through queries because these commands might confuse the internal tracking of server transactions performed by the Microsoft Jet database engine. Instead, use the BeginTrans, CommitTrans, and Rollback methods. Jet translates these methods into the appropriate server commands.

When you work with local data, each Workspace object represents an isolated transaction space. However, this isn't the case for remote data accessed with the Workspace object. You can set the IsolateODBCTrans property on the Workspace object to True to force a Workspace object to have a distinct remote transaction space. This prevents the Workspace object from sharing connections with other Workspace objects, thus guaranteeing transaction isolation. You need to do this only if you use multiple concurrent transactions on your server.

Handling Errors from the Server

Whenever your application causes an error to occur on the server, Microsoft Jet displays two messages: first the Jet engine error message and then the message from the server. If you trap an ODBC error (numbered 3146 through 3299) in a Visual Basic procedure, the Err object returns both the Microsoft Jet error and all server errors concatenated together. By parsing the value returned by Err, you can find out which server error occurred as well as the text of the server error message.

The following error-handling code demonstrates how to obtain the server error number and message when an ODBC error occurs:

```
Dim intRemoteError As Integer, strErrorText As String
Dim intNumStart As Integer, intTextStart As Integer
```

```
Select Case Err

   Case 3146 To 3299
      intNumStart = InStr(Err, "(#") + 2
      intRemoteError = Val(Mid$(Err, intNumStart))
      intTextStart = InStr(Err, "] ") + 2
      strErrorText = Mid$(Err, intTextStart, _
      intNumStart - intTextStart - 3)

      ' Handle remote error.

   Case Else

      ' Handle other errors.

End Select
```

Windows Registry Settings

When you design a client/server application, you can change its performance characteristics by adjusting the following entries in the ...\Jet\3.5\Engines\ODBC folder in the Windows registry. The three dots (...) in the path shown previously represent the location in the Windows Registry that contains the subtree Jet\3.5\Engines\ODBC.

Key	Type	Default	Description
LoginTimeout	Integer	20	Number of seconds to wait for a login attempt to succeed.
QueryTimeout	Integer	60	Number of seconds to wait for a query to execute before failing.
ConnectionTimeout	Integer	600	Number of seconds to wait before closing an idle connection.
AsyncRetryInterval	Integer	500	Number of milliseconds to wait between retries on an asynchronous connection.
AttachCaseSensitive	Boolean	False	If False, will attach to the first name matching the specified string in a match that isn't case-sensitive. If true, requires a case-sensitive match of the name.
AttachableObjects	String	TABLE	The list of objects that Microsoft will allow attachments to (TABLE, NEW, SYSTEMTABLE, ALIAS, SYNONYM).
SnapshotOnly	Boolean	False	If False, you will get index information on attachments to allow dynasets if possible. If index information will be ignored and thereby force snapshots on all attachments

Key	Type	Default	Description
TraceSQLMode	Integer	0	Initiates sending a trace of SQL statements sent to an ODBC data source to the file Sqlout.txt. Values are 0 (No) and 1 (Yes). This entry is interchangeable with SQLTracemode.
TraceODBCAPI	Integer	0	Initiates sending a trace of ODBC API calls to the file Odbcapi.txt. Values are 0 (No) and 1 (Yes).
DisableAsync	Integer	0	Forces synchronous query execution. Values are 0 (use asynchronous query execution if possible) and 1 (force synchronous query execution).
JetTryAuth	Integer	0	Uses the Jet database engine user name and password to log onto the server before prompting. Values are 0 (No) and 1 (Yes). The default is 0.
PreparedInsert	Integer	0	Uses a prepared INSERT statement that inserts data in all columns. Values are 0 (use a custom INSERT statement that inserts only on Null values) and 1 (use a prepared INSERT statement). Using prepared INSERT statements can cause Nulls to overwrite server defaults and can cause triggers to execute on columns that weren't inserted explicitly.
PreparedUpdate	Integer	0	Uses a prepared UPDATE statement that updates data in all columns. Values are 0 (use a custom UPDATE statement that sets only columns that have changed) and 1 (use a prepared UPDATE statement).
FastRequery	Integer	0	Uses a prepared SELECT statement for parameterized queries. Values are 0 (No) and 1 (Yes).

Index

A

action queries, multiple result sets 283
Administer permission in replicated databases 139
aggregate functions, SQL 97
ANSI SQL vs. Microsoft Jet 120
ANSI-89 Level 1 compliant SQL 120
applications
  accessing remote data sources 164, 215
  case sensitive data sources 165
  client/server concepts 239
  creating database applications 13
  database replication 123
  identifying users in multiuser environment 237
  locking records in multiuser environment 230
  making available to multiple users 217
  Microsoft Jet multiuser model 220
  Microsoft Jet supported data sources 163
  multiuser environment 217
  placement of database components 218–9
  Replication Manager 126
  space considerations for temporary indexes 216
  updating user's view with current data 235
  Visual Basic sample applications 3
arrays, copying records to 75
asynchronous connections 269
asynchronous processing 74
asynchronous queries 243, 267, 270
auto-commit mode 294

B

background population 293
backing up records from one database to another 278
batch updates 243, 274
Biblio.mdb code example 23
Biblio.mdb sample database 17
bidirectional synchronization 140
binding, DataBindings collection 88
BOF property, determining recordset position 49–50
bookmarks
  determining support in recordsets 62
  marking record position 60
  scope 60
  vs. record numbers 60
bound controls
  adding multiple bound properties 88
  code example on custom data-bound controls 87
  creating ActiveX controls 87
  in Professional and Enterprise editions 87
  incorporating Visual Basic controls 85
  vs. unbound controls for displaying data 89

C

caching remote data 293
Cartesian product 105
case sensitive data sources 165
chunks, handling of large value fields 81
classes, DAO hierarchy 10
client/server applications
  accessing ODBC data sources with DAO 247
  accessing remote servers 246
  advantage of transactions 294
  advantages of client/server 241
  basic concepts 239
  batch optimistic updates 274
  database engine location 9
  design considerations 293
  direct access through OpenDatabase method 258
  field properties 245
  moving data to server 243
  ODBC data source manager 247
  pass-through queries in DAO 253
  performance considerations 285
  server-based stored procedures 256
  validation rules 244
  vs. file servers 239
  Windows registry settings 299

cloning recordsets 90
collections
  code example on displaying TableDef objects 234
  DAO hierarchy 10
  defaults 12
columns
  aliasing column names 104
  manipulating with ALTER TABLE statement 98
  terminology in relational databases 94
compacting a database 29
compacting databases 32
compacting replicated databases 160
comparison operators in SQL statements 96
complex optimizable expressions 119
complex queries with SELECT statements 112
connection string syntax for external data sources 171
connections
  accessing remote servers 246
  balancing connection use and performance 262
  code example on asynchronous query 270
  code example on executing query on Connection object 273
  code example on OpenConnection method 268
  connect string for ODBC data source 248
  creating in ODBCDirect workspace 267
  DBEngine properties for ODBCDirect 268
  default idle time 261
  limit on asynchronous operations 271
  OpenConnection method 267
  opening asynchronously 269
  reducing instances 260
  releasing 261
  resource management 259
  troubleshooting for external data sources 215
CONSTRAINT clause 100
controls
  bound controls in Professional and Enterprise editions 87
  visual interface 85
Crosstab queries 113–4
CSV format in remote data access 198
currency format for importing into Microsoft Jet 200
cursors
  limitations 283
  lock type compatibility 282
  Microsoft Jet cursor model 280
  retrieving multiple result sets 283
  types in Microsoft SQL Server version 6.0 280

D

DAO
  accessing ODBC data sources 241, 247
  backward compatibility 6
  calling stored procedures 257
  changing database structure programmatically 27
  client/server concepts 239
  creating and modifying databases 7
  data access object model 9
  Data Definition Language (DDL) 13, 97–101
  developing client/server applications 239
  Find methods 59
  forcing collection update 234
  in client/server applications 245
  introduction to data access objects 3
  Jet database engine 4
  linking non-Microsoft Jet database 41
  methods and properties 35
  naming conventions 245
  new features in version 5.0 6
  pass-through queries 253
  refreshing collections 234
  replicating a database 127
  Replication Manager application 126
  resolving synchronization conflicts 152
  workspaces 266
data access
  basic programming principles 13
  bound and unbound controls together 89
  bound controls in Professional and Enterprise editions 87
  capabilities in Visual Basic editions 6
  creating and modifying databases 7
  DAO object model 9
  Data Definition Language (DDL) 13, 97–101
  data manipulation 13
  Data Manipulation Language (DML) 13, 102–120
  displaying recordset values with unbound controls 89
  divisions of database programming 13
  features in Professional and Enterpise editions 85
  information sources 2
  introduction to data access objects 3
  ISAM driver for external data source 79
  Jet database engine 4
  new features in version 3.5 of the Jet database engine 6
  object class hierarchy 10

data access (*continued*)
object oriented programming 12
programming basics 3
sample applications 3
viewing object names and properties 30
Visual Basic database architecture 7–9
Data control
code example on cloning a recordset object 90
data access objects 85–6
Data Definition Language (DDL) 13, 97–101
Data Manipulation Language (DML) 13, 102–120
data pipelining 278–9
data sources
code example on registering 248
ODBCDirect default value 40
transactions on 73
data types
converting from dBASE to Microsoft Jet 177
currency 80
field entries in Schema.ini 199
floating point data in Microsoft Jet 264
GUID 80
Long 80
mapping in Microsoft Jet 264–5
Type property settings for field objects 78–81
data validation 83
database
deleting fields 28
deleting tables 27
database programming divisions 13
database replication
advantages 123
AutoNumber field order 138
backup considerations 159
backups vs. replication 159
binding list boxes to tables 161
BLOBs (binary large objects) 135
closing tables before synchronizing 161
code example on conversion to Design Master 131
code example on error handling 128–9
code example on making new copies 132
code example on replica filters 147
code example on resolving conflicts 152
code example on URL address 143
compacting before synchronizing 160
controlling replica creation 157–8
correcting synchronization errors 155
creating partial replica 146
dbRepSyncInternet constant 144
database replication (*continued*)
described 123
design considerations 157–61
DesignMasterID property 134
discarding records 160
duplicate keys 153
error handling 150–7
excluding information 128
FTP folder in indirect synchronizations 144
generation identifier 135
GUID 135
implementing star topology 140–2
limitations in table size 137
locked objects 133
locked records 155
MakeReplica method 133
moving Design Master to replica 159
new system fields 134
orphaned records 149
partial replication 145–6
primary key 154
read-only replicas 133
referential integrity 153–5
referential integrity errors 154
referential integrity in replication 148
replica filters 148
replica set topology 125
ReplicaID property 134
replicating new objects 130
replicating with DAO 127
replication for multiple users 123
Replication ID field 154
Replication Manager application 126
repopulating partial replicas 149–50
security 139
setting relations 148
size increase 137
size supported by Microsoft Jet 138
solving design errors 156–7
sources of synchronization errors 153–5
star topology 125–6
synchronization conflicts 151–2
synchronized replicas 126
synchronizing 139–45
synchronizing before changes 161
synchronizing data over Internet 143–5
synchronizing in WANs 142–3
synchronizing multiple replicas 145
table-level validation (TLV) 153

database replication (*continued*)
tables added during conversion 136–7
tracking modifications 134–9
UUID (Universally Unique Identifier) 135
databases
32-bit database engine 7
adding fields to tables 28
adding indexes 29
adding tables 27
advantages of client/server 241
alternatives for creating 19
assigning Recordset objects 86
attaching tables from external databases 26
avoiding redundancy 18
basic SQL concepts 93
case sensitive data sources 165
code example on creating Biblio.mdb 23
code example on transactions 71
compacting 32–3
compaction syntax 33
CREATE TABLE statement 97
creating and modifying 7–34
creating indexes 98
creating indexes and relations 21
creating new Jet database 20
DAO hierarchy 10
DAO methods and properties 35
data access objects 85
data store files 8
database engine location 9
database programming in Visual Basic 4
dBASE support in Microsoft Jet 175–9
DDL operations in SQL 97–101
deleting indexes 29
design basics 17
eliminating duplication 16
encryption 32
foreign keys 98, 100
handling recordset locking problems 225
IISAM support for Paradox databases 184
index keys 101
intermediate changes buffer 72
introduction to data access objects 3
ISAM format support 163
keyed fields 15
large value fields 80
linking data in desktop databases 174
linking through Microsoft Jet 246
local and remote 8
databases (*continued*)
maintenance tasks 29
making available to multiple users 217–38
many-to-many relation 15
mapping 30
Microsoft Jet supported data sources 163
modifying structure programmatically 27
multiuser design issues 233–5
ODBC databases 5
opening external tables 169
opening in either shared or exclusive mode 223
opening in read-only mode 224
opening mailbox as a Database object 209
opening through OpenDatabase method 223
optimizing queries 118
placement of database components in multiuser environment 218–9
recognized through DAO and Jet engine 5
referential integrity 22
refreshing DAO collections 234
relational database design 14
relational database model 14
relational database terminology 14
remote vs. client/server database 9
repairing 29, 33
retrieving records 76
Sequential Access Method (ISAM) databases 5
shared vs. exclusive mode in Microsoft Jet 222
space considerations for temporary indexes 216
SQL language components 94
table relations 15
updating user's view with current data 235
Visual Basic database architecture 7–9
DataBindings collection 88
dates
data type handling in Microsoft Jet SQL and Transact-SQL 255
date literal format 106
format for importing into Microsoft Jet 200
dBASE databases 175–9
converting data types to Microsoft Jet 177
deleting records 177
index files 178
memo fields 179
opening directly 177
support in Microsoft Jet Xbase IISAM driver 175
dbRunAsync option 269
DDL and DML commands 95

DDL operations in SQL 97
DELETE query 109
deleting
  links to external data sources 214
  records from dBASE tables 177
Design Master replication 159
disk drives, compacting a database to save space 32
DISTINCT keyword 104
dynamic-type recordset objects 264
dynasets
  connections required 260
  dbOpenDynaset 262
  defined 35
  determining field origin 83
  finding records 54
  performance considerations 291

E

encrypted database 32
error codes for page locking 231
error handling
  code example on obtaining server error number 298
  from servers 298
  generic procedures for recordset locking problems 225
  testing record locking status 232
exporting data into delimited text files 202
external data sources 163–216
  access methods 167
  accessing through ISAM 163
  active components 218
  code example on layout for a fixed-width file 202
  code example on opening 169
  connection string syntax 171
  deleting links 214
  described 5
  importing data formats into Microsoft Jet 200
  linking comma-delimited files 203
  linking data in desktop databases 174
  linking vs. importing 167
  linking vs. opening directly 169
  linking with Lotus worksheet 195
  Microsoft FoxPro databases 163, 179–83
  Microsoft Jet supported data sources 163
  optimizing queries 118
  Paradox databases 163
  passwords 173
external data sources (*continued*)
  paths 171
  performance tips 214
  placement of database components 218–9
  solutions to connection problems 215
  source and connect values 169
  source database types 172
  space considerations for temporary indexes 216
  specifying read-only or write-only access 224
  specifying tables 103
  static components 218
  transaction support 166
  troubleshooting tips 214

F

fields
  copying from recordsets 75
  criteria defining properties 83
  data types 78
  data validation 83
  deleting from database 28
  displaying on request 290
  large value binary data 80
  limits in database replication 138
  random order in replicated databases 138
  reading and writing data 81
  referencing field objects 77
  referencing in a loop 78
  specifying field names with HDR parameter 207
  tracing the origin 83
file names in Schema.ini files 198
Find methods in DAO 57
floating point data in Microsoft Jet 264
foreign keys in referencing tables 22
forms, performance considerations in database applications 289
forward-only-type recordset objects 214, 264
FoxPro databases 179–83
  code example on linking 168, 180
  code example on opening directly 181
  code example on updating lost link 213
  external data 163
  Internet connection 171
  linking with Microsoft Jet 180
  memo fields 183
  Microsoft Jet Xbase IISAM driver support 179
FTP servers, data source names 171

G

graphics, embedding into HTML tables 208
GROUP BY clause 107
GUID data type 80
GUID in database replication 135

H

HAVING clause 107
HDR parameter in connection string 187, 191
Help
  information sources 2
  programming basics 2
heterogeneous joins 288
hierarchy of DAO classes 10
HTML tables
  cells spanning multiple columns 207
  code example on specifying FTP location 205
  described 204
  embedded images and hyperlinks 208
  embedded lists in table cells 208
  IISAM data type interpretation 206
  linking with Microsoft Jet 204
  naming conventions 206
  OneTablePerFile registry setting 209
  opening directly 205
  referencing tables 206
  specifying data types in Schema.ini file 207
  Windows Registry entries for importing text 209
hyperlinks, importing graphic references 208

I

icons, embedding into HTML tables 208
IISAM data sources, HTML tables 163
IISAM drivers
  accessing external data 163
  accessing Microsoft Excel 8.0 192
  accessing spreadsheet information 187
  ANSI and OEM character sets 200
  complex tables in HTML tables 207
  data importing considerations 167
  data type interpretation in HTML tables 206
  dBASE support 175–9
  embedded lists in HTML tables 208
  FoxPro support 179
  HDR parameter 187
  limitations in Microsoft Excel 188
IISAM drivers (*continued*)
  linking text files 196
  Lotus worksheet support 193
  maintaining links to external tables 213
  Microsoft Excel support 188
  opening HTML tables 205
  opening mailbox as a Database object 209
  opening Microsoft Exchange folders 211
  overriding range setting in Windows Registry 187
  Paradox database support 184
  performance tips 214
  Schema.ini files 197
  setting mailbox password 210
  spanning multiple columns in HTML tables 207
  specifying data format for Microsoft Jet 200
  specifying database names 171
  specifying external data sources 169
  specifying field names in delimited text files 198
  specifying text file format in Schema.ini 198
  transaction support 166
  unsupported objects and methods 166
  Windows registry settings 164, 172, 209
importing data formats into Microsoft Jet 200
indexed sequential access method (ISAM) 14
indexes
  adding to database 29
  database keys 15
  deleting from database 29
  in client/server applications 244
  remote index joins 288
  space considerations in Microsoft Jet 216
  SQL CONSTRAINT clause 98
  unique keys 101
inner joins 115
INSERT INTO statement 110
installing ODBC drivers 243
Internet
  exchange argument in database replication 144
  FTP folder in indirect synchronizations 144
  FTP location of external database 171
  HTML file described 204
  opening HTML tables 204
  opening HTML tables with OpenDatabase method 205
  Synchronizer message file 144
  synchronizing data in WANs 143
ISAM databases 73
ISAM drivers, accessing external data 79

J

Jet database engine version 3.5 6
Jet SQL 120–1
joins 114–7
  nested joins 117
  remote index joins 288
  types 114

L

large value fields 80
linking
  comma-delimited files 203
  dBASE tables to Microsoft Jet 176
  deleting links to Microsoft Jet 214
  external and local Jet databases 174–5
  FoxPro tables to Microsoft Jet 180
  maintaining links to external tables 213
  Microsoft Exchange folders with Microsoft Jet database 210
  Paradox databases 184
  refreshing links to tables after moving 213
  SQL views through Microsoft Jet 247
  tables in Microsoft Jet database 168
  vs. importing in external databases 167
list boxes, binding to replicated tables 161
lists, embedded in HTML tables 208
locked records in replicated databases 155
locking
  checking code for page locking errors 231
  code example on pessimistic locking 228
  custom scheme 231
  handling conflicts in Microsoft Jet 223
  locking information (.ldb) file 221
  locktype parameters 282
  optimistic locking considerations 227
  pessimistic locking 226
  pessimistic locking considerations 226
  records being edited 226
  recordset locking in Microsoft Jet 222
  shared data in Microsoft Jet 222
  testing locking status 232
logical operators in SQL statements 96
Lotus worksheets 193–6
  DATABASE and source arguments 194
  IISAM driver support 193
  linking with Microsoft Jet database 195
  opening directly 196

M

many-to-many relationships 15
mapping a database 30
Masked Edit control 87
.mdb files 8
Memo field size 80
Memo fields, limits in database replication 137
memory, handling of large value fields 81
Microsoft Access
  accessing Microsoft Exchange folder 212
  AutoNumber field in client/server applications 244
  complications in client/server applications 244
  porting to SQL Server database 4
  preventing exclusive mode 222
  Replication Manager application 126
Microsoft Excel 187–93
  accessing through IISAM drivers 187
  DATABASE parameter in remote access 188
  IISAM driver support and limitations 188
  importing recordsets 77
  linking with Microsoft Jet database 190
  opening from Microsoft Jet database 190
  specifying first row as header or record 191
Microsoft Exchange
  accessing folders from Microsoft Access 212
  accessing mailbox as a Database object 209
  connection string keyword MAPILEVEL= 210
  creating and deleting folders as Database objects 212
  described 209
  IISAM access restrictions 212
  linking folders with Microsoft Jet database 210
  opening folders directly through OpenDatabase method 211
Microsoft Jet
  32-bit database engine 7
  access limitations in text files 196
  accessing ODBC data sources 241, 242, 246
  attaching from external databases 26
  auto-commit mode 294
  code example on linking HTML table 204
  code example on linking tables 174
  code example on opening Microsoft Excel 7.0 workbook 192
  complex optimizable expressions 119
  connection management 259
  connection timeout 261
  controlling background population 293

Microsoft Jet (*continued*)
converting dBASE data types 177
creating audit trail 237
custom locking schemes 231
DAO programming model 7
data access capabilities in the Learning Edition 6
database engine location 9
database engine principles 8
database replication size supported 138
databases recognized through DAO and Jet engine 5
date data type handling 255
date literal format 106
dBASE support 175–9
described 7
determining data type in fields 199
developing client/server applications 239
enhanced features 121
error handling from servers 298
external data sources 163
file servers vs. client/server 239
floating point data 264
foreign keys in referencing tables 22
heterogeneous joins 288
information sources 2
introduction to data access objects 3
Jet database engine 4
linking comma-delimited files 203
linking dBASE tables 176
linking SQL views 247
linking with delimited text files 196
linking with Lotus worksheet 195
locking information (.ldb) file 221
locking records in multiuser applications 230
locking shared data in recordsets 222
mapping data types 264
Microsoft Excel support 188
migrating database objects 273
multiuser model 220
multiuser settings in Windows registry 221
navigational and relational models 94
new features in version 3.5 of the Jet database engine 6
optimizing expressions 287
page locking 226–33
performance tips 214
placement of database components 218–9
procedure for creating a new Jet database 20
processing multiple result sets 253
Microsoft Jet (*continued*)
Rushmore technology 119
schema 13
shared vs. exclusive mode 222
space considerations for temporary indexes 216
specifying data format in Schema.ini 200
specifying read-only or write-only access 224
supported capabilities vs. ODBCDirect 242
supported networks 221
transactions in multiuser environment 236
updating dBASE indexes 178
user interface 8
validation rules 244
vs. ANSI SQL 120
vs. ODBC data bases 70
vs. ODBCDirect 242
WHERE clause 105
workgroup information file 220
Microsoft SQL Server ODBC databases 5
modem connections
partial database replication 146
updating information in WANs 142
multiuser database design 233
multiuser model in Microsoft Jet 220

N

naming conventions in client/server applications 245
navigational database model 94
nested joins 117
nested subqueries 112
nested transaction limit in database replication 138
nested transactions 73
network hardware
100 MB Ethernet standard 238
optimizing performance 238
network servers
hardware considerations 238
making applications available to multiple users 217
networks support in Microsoft Jet 221
normalization in relational databases 16

O

object variables, code example on creating a new database 20
objects, DAO class hierarchy 10
ODBC data source manager 247

ODBC data sources
access methods 167
accessing with DAO 3, 241, 247
accessing with Microsoft Jet 242, 246
accessing with ODBCDirect 243
code example on linking tables 250
connect strings 248
connecting to 259
data source name (DSN) 171
database engine location 9
described 5
linking tables 245
saving passwords between sessions 173
Windows registry settings 172
ODBC databases
attaching tables 26
connecting to 267
described 5
vs. Microsoft Jet 70
ODBC drivers, installing 243
ODBC Refresh Interval setting 262
ODBCDirect 243
accessing ODBC data sources 243
advantages 243
asynchronous queries 270
bypassing Microsoft Jet 241
capabilities not present 242
code example on CacheSize method *** fix 272
code example on creating a stored procedure 284
Connections collection 265
CreateWorkspace method 266
DBEngine properties 268
forward-scrolling-type Recordset 283
switching between connection and database objects 273
vs. Microsfot Jet 242
ODBCDirect data source default value 40
Open Database Connectivity (ODBC) data sources 9
optimistic batch updates 274
optimization
BLOB in database replication 135
connection-related performance issues 262
design of client/server application 285
form location considerations 290
forward-only-type Recordset objects 214
handling of large value fields 81
hardware considerations 238
optimization (*continued*)
linking databases vs. opening directly 247
Microsoft Jet performance tips 214
multiple-ported design 238
multiple-table queries vs. subforms 291
optimizable expressions 119
optimizing Microsoft Jet specific expressions 287
performance considerations for forms 289
performance enhancements in Microsoft Jet 3.5 74
queries 118–9
record locking strategies 230
referencing Field objects 78
remote data caching 293
Rushmore technology 119
snapshots vs. dynasets 291
strategies for database queries 118
ORDER BY clause 108
orphaned records in database replication 149
outer joins 116

P

page locking in Microsoft Jet 226–33
Paradox databases 184–7
code example on linking with Microsoft Jet 184
converting data types to Microsoft Jet 185
IISAM driver support 183
index files 186
net path settings 186
opening directly 185
parameter queries 63–4, 65
pass-through queries 251–6
advantages 251
code example 254
creating in DAO 253
executing without QueryDefs 256
in stored procedures 256
passwords
accessing mailbox as a Database object 210
accessing remote data sources 173
identifying users 237
saving between ODBC sessions 173
permissions
controlling replica creation 157
database security 139
populating partial database replicas 149
porting Microsoft Access to SQL Server database 4

programming
code example on OpenDatabase method 12
database programming in Visual Basic 4
divisions of database programming 13
help sources 2
ProgressBar control 87
properties in client/server applications 245
pseudo index 247

Q

queries
code example on pass-through query 254
locally evaluated 286
multiple-table queries vs. subforms 291
opening recordset objects asynchronously 272
optimizing 118
pass-through queries in DAO 253
speeding up 286
splitting WHERE clause restrictions 287
SQL pass-through queries 251

R

read-only access in Microsoft Jet 224
records
adding with INSERT INTO statement 110
deleting with DELETE queries 109
extracting data 75
finding a record using Seek method 54
in relational databases 14
large value fields 80
recordsets
accessing mailbox as a Database object 210
adding records 68–70
aliasing column names 104
assigning Recordset objects to database 86
background population 293
bookmarks support 62
caching recordset objects 293
choosing cursor type 280
choosing type 262
cloning 90
code example on BOF and EOF properties 49
code example on bookmarking records 60
code example on creating audit trail 237
code example on deleting records 67
code example on indexed table 42
code example on progress bar 53
recordsets (*continued*)
code example on Requery method 45
code example on requerying 235
collision handling in batch updates 276
connections required 260
copying entire records to an array 75
copying single fields 75
counting number of records 51
creating from base table 41
creating from text file table 204
creating variables 37
current position with BOF 49
cursor limitations 283
data type considerations 79
data validation 83
default types 38
defined 35
deleting records 67–8
described 35
detecting limits 49
determining field origin 83
displaying values with unbound controls 89
dynamic-type recordset objects 264
dynasets vs. snapshots 291
error handling on locking problems 225
field order in Fields collection 77
Fields collection 77
Find methods 58
finding current position 52
finding records with Seek method 55
finding specific records 54
forward-only-type Recordset objects 214, 264
implementing locking 223
importing to Microsoft Excel 77
inconsistent dynasets 66
linking Microsoft Exchange folders with Microsoft Jet database 210
linking non-Microsoft Jet database 41
linking tables in remote databases 245
locating records in dynasets or snapshots 57
locking methods 227
locking shared data in Microsoft Jet 222
locking 282
marking position with bookmarks 60
maximum record size 81
Microsoft Jet cursor model 280
modifying existing records 65–7
moving through 47

recordsets (*continued*)
moving to last modified record 69
opening asynchronously 272
optimistic batch updates 274
options for locking 40
options for opening 39
page locking 226–33
parameter queries 63–4
performance considerations for forms 289
pessimistic locking 226
reading and writing data 81
reading tabular HTLM information 205
record count in multiuser environment 52
record position 52
recreating queries 45
referencing field objects 77
releasing connections 261
requerying data in multiuser environment 235
retrieving records in batches 69
running stored queries 44
Seek method comparison strings 55
Seek method 42
snapshot-type 263
sorting 44
source argument 38
specifying external data sources 169
specifying read-only or write-only access 224
testing locking status 232
timestamp 292
tracking changes 70
updating records 69
write permissions 81
referential integrity 100
referential integrity in replicated databases 153, 154
registry, Windows settings 299
relational database design 14
relational database structure 14
relational database system hierarchies 9
relational databases 94
remote data access
attaching tables from external databases 26
caching 293
data sources 8
optimizing form design 290
speeding up queries 286
timestamp 292
remote data sources
access methods 167
ANSI and OEM character sets 200
case sensitive data 165
connection considerations 164
delimiter in text files 198
HDR parameter in connection string 191
Lotus worksheets 194
Microsoft Jet support 163
saving passwords 173
specifying column width in fixed-length files 199
unsupported objects and methods 166
remote index joins 288
repairing corrupted databases 33
replicated database information 136
replicated databases 123–61
Replication Manager 126
replication topology 125
result sets, processing multiple result sets 253
RichTextBox control 87
rows, terminology in relational databases 94
Rushmore technology 119–20

S

sample applications 3
Schema.ini files 197, 202
security
accessing remote data sources 173
creating audit trail 237
identifying users 237
in client/server applications 245
model in replicated databases 139
restricting access to records 146
Seek method
code example on searching keyed records 56
comparison operators 57
comparison strings 55
finding records in recordsets 55
SELECT INTO statement 109
Sequential Access Method (ISAM) databases 5
server-based stored procedures 256
servers
advantages of client/server 241
connect string for ODBC data source 248
controlling background population 293

servers (*continued*)
direct access through OpenDatabase method 258
handling errors from 298
moving data to server in client/server applications 244
shared vs. exclusive mode in Microsoft Jet 222
snapshots
defined 35
performance considerations 291
sorting case sensitive data sources 165
spreadsheets, accessing through IISAM drivers 187
SQL
ANSI-89 Level 1 120
clauses 95
DDL and DML commands 95
DDL operations 97–101
DML operations 102–120
language components 94
Microsoft Jet vs. ANSI SQL 120
operators 96
SQL queries 93–121
basic concepts 93
calling stored procedures 257
creating a delete query 109
crosstab queries 113
described 93
DISTINCT keyword 104
filtering results 104
join types 114
navigational methods 94
nested joins 117
optimization strategies 118
outer joins 116
pass-through queries 251
processing multiple result sets 253
pseudo index 247
SELECT statement 102
sorting results 104
splitting WHERE clause restrictions 287
TRANSFORM statement 113, 121
UPDATE queries 111
SQL statements
aggregate functions 97
aliasing column names 104
ALTER TABLE statement 98, 100
ANSI SQL vs. Microsoft Jet 120
auto-commit mode 294
Cartesian product 105
changing database structure programmatically 27
SQL statements (*continued*)
code example on OpenRecordset method 43
code example on recordset sort 44
code example on Requery method 45
code example on stored parameter query 64
code example progress bar 53
comparison operators 96
complex queries 112
CONSTRAINT clause 98, 100
CREATE INDEX statement 98, 99
CREATE TABLE statement 98
creating a new table with SELECT INTO 109
creating tables 97
date literal format 106
DD2L statements 95
DELETE queries 109
DISTINCT keyword 104
DISTINCTROW predicate 116
DML statements 95
file servers vs. client/server 239
GROUP BY clause 107
HAVING clause 107
inner joins 115
INSERT INTO statement 110
join types 114
locally evaluated queries 286
logical operators 96
merging results into single UNION operation 118
multiple result sets 283
nested joins 117
optimizable expressions 119
ORDER BY clause 108
outer joins 116
running stored queries 44
SELECT queries 102
TOP predicate 105
TRANSFORM statement 113
UNION operation 117
UPDATE queries 111
using Visual Basic variables 104
WHERE clause 105
WITH OWNERACCESS OPTION 109
star topology synchronization 140
stored procedures
code example on creating 284
handling return values 257
pass-through queries 256
Structured Query Language (SQL) 14
subqueries nested inside SELECT statements 112

supported networks in Microsoft Jet 221
synchronization conflicts in database replication 151
synchronization errors in replicated databases 150, 153
synchronized database versions 123
synchronizing database replicas 139, 145, 161
synchronous and asynchronous processing 74

T

table-level validation (TLV) in replicated databases 153
tables
  adding fields to database tables 28
  adding to database 27
  attaching from external databases 26
  binding list boxes in replicated databases 161
  closing before synchronizing 161
  combinations in HTML tables 207
  CREATE TABLE statement 97
  deleting from database 27
  determining field origin 83
  embedded lists in HTML 208
  in HTML 204
  including large value fields 81
  keeping local in database replication 130
  linking in Microsoft Jet database 168–9
  linking in remote databases 245
  linking through Microsoft Jet 246
  manipulating with ALTER TABLE statement 98
  naming conventions in HTML tables 206
  normalization in relational databases 16
  one-to-many relation 15
  opening external databases 169
  opening external Microsoft Jet tables directly 175
  optimizing access to large external tables 214
  refreshing links after moving 213
  relationships in database replication 146
table-type recordset objects 262
temporary database 72
text files
  access limitations 196
  opening directly through OpenDatabase method 204
timeouts, default idle time 261
timestamp 292
TOP predicate 105
transactions
  advantages 294
  auto-commit mode 294
transactions (*continued*)
  basic syntax 71
  bulk insertions 296
  code example on bulk updates 295
  defined 70
  effect of open time on multiuser concurrency 237
  IISAM support 166
  limit of nested in database replication 138
  limitations 298
  managing in Microsoft Jet 72
  Microsoft Jet vs. ODBC data bases 70
  nesting 73
  on external data sources 73
  performance considerations 74
  pessimistic locking 226
  rollback 73
  running asynchronous queries 272
  synchronous and asynchronous processing 74
  updating data in multiuser environment 236
TRANSFORM statement 113
trappable errors
  generic procedures for recordset locking problems 225
Type property settings for field objects 78

U

union queries 117
updating
  advantage of transactions 294
  bulk updates in Microsoft Jet 295
  code example on data pipelining 279
  code example on forcing changes 277
  code example on optimistic batch update 274
  collision handling 276
  data pipelining 278
  database with UPDATE query 111
  discarding records in replicated databases 160
  inconsistent dynasets 66
  information in WANs 142
  links to tables after moving 213
  optimistic batch updates 274
  performance considerations 292
  records in recordsets 69
  refreshing collections in multiuser environment 234
  synchronous and asynchronous processing 74
  write permissions 81
URL address for synchronizing data 143

user interface
controlling replica creation 157
incorporating Visual Basic controls 85
Jet database engine 8

V

validation rules 83, 244
variables
declaring type Recordset 37
using VB variables in SQL statements 104
Visual Basic database architecture 7
Visual Basic editions 6
Visual Basic Enterprise Edition data access capabilities 6
Visual Basic Learning Edition data access capabilities 6
Visual Basic Professional Edition data access capabilities 6

W

WAN synchronization strategies 142, 143
WHERE clause 105
WHERE clause restrictions 287
wildcards
ANSI SQL vs. Microsoft Jet 120
Windows registry
connection settings in client/server applications 299
Microsoft Jet multiuser settings 221
Windows registry settings
for IISAM drivers 164
HTML IISAM entries 209
overriding IISAM settings 187
WITH OWNERACCESS OPTION 109
workgroup information file in Microsoft Jet 220
Workspace object 266

Crystal Reports™ for Visual Basic User's Manual

Microsoft® Visual Basic®

Version 5.0
Programming System for Windows® 95 and Windows NT®

Microsoft Corporation

Information in this document is subject to change without notice. Companies, names, and data used in examples herein are fictitious unless otherwise noted. Complying with all applicable copyright laws is the responsibility of the user. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. The furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 1996 Microsoft Corporation. All rights reserved.

Microsoft® Visual Basic®
Programming System for Windows® 95 and Windows NT®, Version 5.0

Microsoft, FoxPro, Microsoft Press, Microsoft QuickBasic, MS, MS-DOS, Visual Basic, Visual C++, Windows, and Windows NT are registered trademarks of Microsoft Corporation in the USA and other countries.

Crystal Reports is a registered trademark of Seagate Software Information Management Group, Inc.

Other product and company names mentioned herein may be the trademarks of their respective owners.

Document No. DD93015-1296

Contents

Chapter 1 Configuring Crystal Reports to work like you do 1

Configuration examples 1
To specify a default data directory 1
To specify the types of databases listed in the Choose Database File dialog box 2
To specify the default view used to preview your report 2
To specify the default fonts for a section of your report 2
To specify a SQL/ODBC server as your primary data source 3
To specify a report template as your primary data source 4
To specify the default settings Crystal Reports uses for formatting fields 4

Chapter 2 Building Reports 5

Getting Started 5
Selecting a Report Type 6
Selecting a data source 6
Linking Database Tables 6
Understanding the Design environment 7
Selecting fields 7
Moving, resizing, formatting and deleting fields 8
Sorting report data 9
Grouping report data 10
Summarizing and totaling 12
Selecting records 13
Creating formulas 15
Adding a graph/chart 15
OLE Objects 16
Adding a report title 16
Previewing your report 17
Saving your report 17
Printing your report 17

Chapter 3 Formulas 19
What are Formulas? 19
How formulas are created - An Introduction to the Formula Editor 21
Other formula conventions 23
Formula Syntax 25
How formulas are evaluated - Order of precedence 27
How to delete formulas from your report 28
How to copy formulas from Crystal Reports online Help 29

Chapter 4 Functions 31
Function Reference 31

Chapter 5 Operators and Variables 57
Operator Reference 57

Chapter 6 The Crystal ActiveX Control 77
The Crystal Report Engine 77
Before Using the ActiveX Control 77
Using the Crystal Report Engine 78
Adding the ActiveX Control to a Project 79
Using the ActiveX Control 80
ActiveX Properties Index 81
ActiveX Methods Index 85
Crystal ActiveX Control Properties 86
Crystal ActiveX Control Methods 129

Appendix A Product Support, Upgrades, and Licensing 131
Product Support 131
Facts On Demand 132
Fax Support - FREE 132
Mail-in Support - FREE 132
Telephone Support 133
Extended Technical Support Policy 134
Product Registration 134
Crystal Reports Upgrade Plan 135
Product Replacement Policy 135
License Agreement And Limited Warranty 136
Restricted Rights Legend 139
International Partners Directory 140

CHAPTER 1

Configuring Crystal Reports to work like you do

Crystal Reports lets you customize many of the program's default settings to fit the way you work. These settings affect such things as:

- your working environment,
- the way you select databases,
- SQL and ODBC access,
- the way various data types are formatted, and
- the fonts you use for fields and text.

To change your default settings, choose the Options command from the File menu. The File Options dialog box appears. Click the tab appropriate to the option you want to change, as in the examples below. With the options feature these changes are easy to make.

Configuration examples

When configuring Crystal Reports you may want to specify the default data directory that is used when creating new reports, and what kinds of databases will be listed in the Choose Database File dialog box whenever it appears. For complete descriptions of all configuration options, search for *File Options command* in Crystal Reports Help.

To specify a default data directory

1 Click the Database Tab in the File Options dialog box. Database options appear.

2 Specify your default directory using one of the following methods:

- Type the full path in the Data Directory box (C:\CRW for example), or
- Click the Browse button, make your selection using the Set Directory dialog box when it appears.

3 Click OK when finished to return to the Database Tab.

To specify the types of databases listed in the Choose Database File dialog box

When you specify the types of databases to be listed in the Choose Database File dialog box, you limit the types of files listed. You may choose to do this when you are only working with one or two types of databases and you want only those types to be available in the File Name scroll box.

1 Click the Database Tab in the File Options dialog box. Database options appear.

2 In the Database Selector edit box, type the filename extensions for the types of databases you want listed in the Choose Database File dialog box.

3 Click OK when finished to return to the Database Tab.

Note When entering filename types (*.db; *.ddf; *.ovd, etc.) in the Database Selector box you must: place an asterisk before the extension, and a semicolon between file types.

To specify the default view used to preview your report

The default view is the view that Crystal Reports will use when you view your report in the Preview Window. After the report has been displayed in the default view, you can select a different view if necessary.

1 Click the Layout Tab in the File Options dialog box. Layout options appear.

2 In the lower left corner of the tab are the Preview Pages options: Fit Size, Fit Width, and Fit Page. Select one of the three choices to specify the default view you want Crystal Reports to use when you view pages in the Preview Window.

3 Click OK to return to the program.

Now each time you view a report in the Preview Window, Crystal Reports will start with the option you selected.

To specify the default fonts for a section of your report

1 Click the Font Tab in the File Options dialog box. Font options appear.

2 The Font Tab has two vertical rows of buttons; one row for changing Default Field Fonts, and the other row for changing Default Text Fonts. Click the button corresponding to the section you wish to change the default font for. The Font dialog box appears.

3 Select the desired font, style and point size to be used as the default, and click OK when finished to return to the Font Tab.

- For each section that you want to change the default font for, repeat steps 2 and 3 above.

4 Click OK again to return to the program.

Now each time you use a Field Font or a Text Font that you changed the section default for, the font you selected will be used in that section. All the fields and text you place in each section of your report will appear using the default fonts you specified for that section.

To specify a SQL/ODBC server as your primary data source

Specifying a SQL/ODBC server as your primary data source is a four step process:

- set the *Prefer to Create new reports from* SQL Tables option,
- select the Server Type
- select the Skip Server Type Dialog, and
- enter the Server Name, Database, and User ID.

This requires you to work with two of the tabs in the File Options dialog box: New Report, and SQL.

To set the *Prefer to Create new reports from* option

1 Click the New Report Tab in the File Options dialog box. New Report Options appear.

2 Select the SQL Tables option from the *Prefer to create new reports from* section.

To select the Server Type

1 Click the SQL Tab in the File Options dialog box. SQL options appear.

2 Select the appropriate Server Type from the *Server Type* drop down box. The data source appears in the edit box.

To skip the Server Type dialog

- While still in the SQL Tab, toggle the *Skip Server Type dialog* check box On.

To enter the Server Name, Database, and User ID

1 Enter the desired server name into the *Server Name* edit box.

2 Enter the desired database name into the *Database* edit box.

3 Enter the desired User ID into the *User ID* edit box.

4 Click OK to return to the program.

Note If you turn off the Use Report Gallery for new reports option after specifying a SQL/ODBC Server as your primary data source, when you begin a new report you will bypass the Report Gallery and go straight to the SQL dialog box.

To Turn off the *Use Report Gallery for new report* option

1 Click the New Report Tab in the File Options dialog box. New Report options appear.

2 Toggle the *Use Report Gallery for new reports* option Off.

3 Click OK to return to the program.

Note If the check mark is not visible in the checkbox, Use Report Gallery for new reports is already off.

Now when you choose the New Report command from the File menu or click the New Report button on the button bar, you go straight to the dialog box specific to logging on to the server you selected. Simply enter the correct User ID and Password to log onto the server.

To specify a report template as your primary data source

1 Click the New Report Tab in the File Options dialog box. New Report options appear.

2 Toggle the *Prefer to create new reports from Other Reports* option On.

3 Click OK to return to the program.

Now when you choose the New Report command from the File menu or click the New Report button on the button bar, the Select Report Template dialog box appears, listing report templates. Select the report you want to use as a template, and click OK to open the template.

To specify the default settings Crystal Reports uses for formatting fields

These default settings will be used for formatting fields when you begin a new report. You can specify the default settings that you will use for most of your reports.

1 Click the Fields Tab in the File Options dialog box. Field formatting options appear.

2 Select the field type you want to change formatting options for. A dialog box appears with formatting options for the field type.

3 Make the changes you want to the default settings in the dialog box, and click OK to return to the Fields Tab.

- For each field type you want to change default options for, repeat steps 2 and 3 above.

4 Click OK to return to the program.

Now whenever you have any of the fields in your report that you have set formatting options for, they will automatically be printed using that formatting as specified.

CHAPTER 2

Building Reports

Use the following steps as a guide for building reports with Crystal Reports.

Note You must create the reports you intend to have available through your application before you begin working with the Custom Controls. You cannot create reports using the Custom Controls.

Getting Started

1 Open Crystal Reports.

2 Click the New Report button on the button bar. The Report Gallery appears.

- Click one of the Report Creation Expert buttons for step-by-step help in creating a report,
- click the Another Report button to use another report as a template for building your report, or
- click the Custom button to build a custom report.

3 When you click the Custom button, the Report Gallery expands.

Note The Report Gallery will not appear if you choose the New command on the File menu to create a new report. The Report Gallery will only appear when you click the New Report button on the button bar and you have the *Use Report Gallery for new reports* check box toggled On in the New Report Tab of the File Options dialog box.

Selecting a Report Type

- Click the Report Type button for the report you want to create:
  - Click the Custom Report button to create a standard report in a row/column or data block format.
  - Click the Custom Cross-Tab button to create a Cross-Tab report that presents summarized data in a row versus column structure similar to that of a spreadsheet. Cross-Tab reports are easy to read and are useful for making rapid comparisons or identifying trends.
  - Click the Custom Mail Label button to create mailing labels or label-type items.

Selecting a data source

- Click the Data Type button to choose the data source you want to use to create your report:
  - Click the Data File button to activate one or more database tables for use in your report. The Choose Database File dialog box appears. Select a database table to activate for your report and click OK when finished.
  - Click the SQL Table button to open any of the wide variety of SQL data sources that Crystal Reports supports. The Log On Server dialog box appears. Log on to the desired server and then choose the desired SQL tables for use in your report.

Linking Database Tables

If you have selected two or more database tables, you will need to link them. You link database tables so records from one database will match related records from another. To link, use a field that is common to both databases. Crystal Report uses the link to match up records from one database with those from the other.

When you choose multiple tables, the Visual Linking Expert will appear. Use the Visual Linking Expert to link two or more databases together. You can link database tables manually or automatically.

Manually

1. Highlight a field in the database you want to link from that is common to a field in a second database that you want to link to.
2. Drag the field from the first database to the second. If the link is possible, a link line will appear to indicate a successful link.
3. To edit a link, highlight the link line and click the Options button. The Link Options dialog box appears where you can make the desired changes.

Automatically

1 Click the Smart Linking button in the Visual Linking Expert.

2 The Smart Linking feature will create logical links between tables in your report automatically. If links are not possible, a message will be issued.

Understanding the Design environment

The Design Window appears with an empty Title, Page Header, Details, Page Footer and Summary section for use in designing your report. You create your report by inserting and formatting items in each of these sections.

Each report section has its own printing characteristics:

- the Title section prints once, at the beginning of the report,
- the Page Header section prints at the beginning of each page,
- the Details section prints once for each record,
- the Page Footer section prints at the end of each page, and
- the Summary section prints once, at the end of the report.

It is important to understand these characteristics because they affect *when* and *how often* things get printed.

Selecting fields

The Insert Database Field dialog box appears on screen with the Design Window when you open a new report. This dialog box displays a list of fields and tables in the current database available for use in your report. To speed the entry of multiple fields, the box remains on screen until you click the Done button.

- Select the field(s) you want to appear on the report. You can:
  - highlight one field at a time, click the Insert button, and then place the field on your report,
  - highlight one field at a time and drag it onto your report,
  - use the Shift-Click combination to select a number of contiguous fields or the Ctrl-Click combination to select fields from the list at random and drag them onto your report.

Note If you placed multiple fields, they will appear next to each other in the order they appear in the Insert Database Field dialog box.

Crystal Reports marks the position of each field with a rectangular box. The characters in the box indicate whether the field is text (XXX...), number (555...), currency ($555...), date (12/31/99), or Boolean (T/F). The number of characters in the box indicates the number of characters allowed for the field in the database. You can change the appearance of fields by choosing the Options command from the File menu and clicking the *Show Field Names Option* On on the Layout Tab of the File Options dialog box. Each field will now be indicated by its name rather than the corresponding character.

Crystal Reports will automatically place corresponding field titles for each field in the Page Header section directly above the corresponding field. You can disable this option by choosing the Options command from the File menu and clicking the *Insert Field Titles* option Off on the Layout Tab of the File Options dialog box.

Moving, resizing, formatting and deleting fields

1 To move, resize, format, add borders and colors, delete, or insert a subtotal, summary or grand total for a field, you first have to click the field to select it. When you select a field, black handles appear on the right and left sides to indicate it is selected.

Note You can select multiple fields at one time using the Shift-Click method, or by choosing the Select Fields command from the Edit menu. The Select Fields command activates a cross-hair cursor you can use to drag and draw a rectangle to surround the desired field(s), known as the marquee selection technique.

2 Once a field is selected, you then tell Crystal Reports what you want to do with it:

- To move a field, drag the field to the desired position.
- To resize a field, drag the right or left handle until the field is the desired size.
- To format a field (alignment, number, currency, date display, etc.), right-click the field and choose the Change Format command from the shortcut menu. A Format dialog box appears specific to the data type of the selected field. Make the desired changes and click OK when finished.
- To set up borders, background fill and drop shadows, right-click the field and choose the Change Border and Colors command from the shortcut menu. The Format Border and Colors dialog box appears. Make the desired selections and click OK when finished.
- To change the font, right-click the field and choose the Change Font command from the shortcut menu. The Font dialog box appears. Make the desired changes and click OK when finished.
- To delete a selected field, simply press Delete.

Note When you right-click a field, it selects the field and displays a shortcut menu in one single step.

Note Many of the font and formatting options are also available as buttons on the format bar.

Sorting report data

You may want to change the way data is sorted within your report. Sorting means the order in which values are displayed. Information in a phone book, for example, is sorted alphabetically by name to create an order and speed efficiency in accessing the data contained within it.

When you sort in Crystal Reports, the program asks you to select two things: the field you want your sort to be based on (sort field), and the sort direction (Ascending meaning smallest to largest, 1 to 9, A to Z, False to True or Descending meaning largest to smallest, 9 to 1, Z to A, True to False).

A sort field is a field that determines the order in which data appears in your report. You can use any field as a sort field. When you select a sort field, Crystal Reports sorts the records in the report based on the values in the selected field. Your Regional or International settings and the field's data type determine the method in which the data for the field is sorted. For example, if your settings are for the United States:

- **String fields**
  Single character values are sorted so that blanks have the lowest value, then punctuation, then numbers, then uppercase letters, and finally lowercase letters. Then two character values are sorted, then three, etc., using the same rules.
- **Currency fields**
  Currency fields are sorted in numeric order.
- **Number fields**
  Number fields are sorted in numeric order.
- **Date fields**
  Date fields are sorted in chronological order.
- **Boolean Fields**
  Boolean Fields are sorted so that false values (0) come first, then true values (1).

Crystal Reports allows you to perform single field or multiple field sorts. Single field sorts are sorts in which all the records used in the report are sorted based on the values in a single field. In multiple field sorts, Crystal Reports first sorts the records based on the values in the first field selected, putting them in ascending or descending order as specified. When two or more records have the same field value in the first sort field, it then sorts those records (and those alone) based on the value in the second sort field. For example, in a sort based on last name and then first name (in ascending order), "Smith, Bob" would be returned before "Smith, John."

1. Click the Sort Records button on the button bar. The Record Sort Order dialog box appears.
2. Highlight the field(s) to be sorted from the Report Fields list box.
3. Click the Add button. The highlighted field(s) will be added to the Sort fields list box.
4. Specify the sort direction by clicking the corresponding option button. Choose from either Ascending (A to Z, 1 to 9) or Descending (Z to A, 9 to 1).
5. Click OK when finished to return to your report. Crystal Reports will sort the records as specified.

Grouping report data

Crystal Reports enables you to group data in your report. A group is a set of records that are related to each other in some way. For example, in a customer list, a group could consist of all those customers living in the same zip code, or in the same state. In a sales report, a group could consist of all the orders placed by the same customer, or all of the orders generated by a specific sales representative.

Crystal Reports allows you great flexibility in grouping data. It also gives you the ability to create a number of different kinds of group values.

A group value is the value generated as the result of an evaluation, a tally, or a calculation performed on data from a single group. A subtotal is one kind of group value; it is the sum of all of the values from a single field, from all the records in a group. In a sales report, for example, if you subtotal sales by sales representative, Crystal Reports gathers all the records that belong to the sales representative and totals the sales amounts from all the records. In a group average, Crystal Reports averages the values in a group of records; in a group count, it counts the values in a group of records, etc. Group values are important tools for creating powerful reports. Crystal Reports gives you several alternatives for working with group data. It enables you to:

- calculate the sum of values,
- calculate the standard deviation or variance of the values,
- average the values,
- count the values,
- determine the highest value, and
- determine the lowest value in the group.

While there may be many data fields on a report, there is typically only one field for which you are interested in grouping the data. In a sales report, for example, it would probably be the field listing the amount of sales; in a commission report, it would probably be the field listing the amount of commission, etc.

1 When you group data in your report, first select the field you want to group.

2 Once the field is selected, choose the Group Section command from the Insert menu. The Insert Group Section dialog box appears.

3 Select a field (a sort and group by field) to trigger grouping whenever its value changes. In grouping your data by state, for example, you would use the state field to create a new group (and generate a group value if selected) whenever the state changes. Likewise, the ZIP code field would trigger a grouping whenever the ZIP code changes.

Once you have made these simple selections, Crystal Reports groups the data as specified.

Creating Custom Groups

Most of the time you sort and group your data based on the values in a field in your report. For example, if you have a customer list and you want to sort and group it by state, the program first sorts the list by state and then breaks the list into state groups whenever the value in the State field changes.

Sometimes, however, you may not want to group based on the values found in one of the fields on your report.

- Your report may not contain the field you want to group on. For example, your report contains a City field and a State field but no Country field, but you want to group by county.
- Your report may contain the field you want to group on, but you are not happy with the grouping based on the values in that field. For example, you have a Color field on your report that includes specific color names (Logan Green, Sky Blue, Emerald Green, Navy Blue, etc.) but you want all "flavors" of each color to appear as a single group (Greens, Blues, Reds, etc.).
- Your report may contain the field you want to group on, but you want to select specific values or ranges of values for each group. For example, you might want one group to contain records where gross sales are less than a certain value, a second group where gross sales are greater than a certain value, and a final group where gross sales fall between two values.

Crystal Reports provides specified order grouping as a solution to these custom sorting and grouping challenges. Specified order grouping enables you to create the groups you want to appear on your report and the records that each group contains. Your only real limitation is that a record can be assigned to only one group.

To use Specified Order grouping, select in specified order as your sort option whenever the program provides you with that option. The program gives you the in specified order option whenever you create groups using the Group, Subtotal, or Summary command on the Insert menu or whenever you create groups while building a report using one of the Report Creation Experts.

Summarizing and totaling

When Crystal Reports summarizes and totals data, it breaks it into groups and then performs the specified operation on the values in each group. It does this automatically; all you have to do is specify:

- the field you want summarized/totaled,
- the type of operation to be performed on the field,
- the field that is going to trigger a new group whenever its value changes, and
- the sort order.

Inserting a Subtotal

1 To insert a subtotal (for numeric fields only), right-click the field and choose the Insert Subtotal command from the shortcut menu. The Insert Subtotal dialog box appears.

2 Select the field you want to trigger a new subtotal whenever its value changes.

3 Specify the sort direction: Ascending (A to Z, 1 to 9) or Descending (Z to A, 9 to 1).

4 Crystal Reports creates a Group Header and Group Footer section and automatically places the subtotal in the Group Footer section.

Inserting a Grand Total

1 To insert a grand total (or a report average, a report count, etc.) at the end of the report, right-click the field and choose the Insert Grand Total command from the shortcut menu. The Insert Grand Total dialog box appears.

2 Select the desired operation from the drop down list and click OK when finished.

3 Crystal Reports creates a Grand Total section to hold the grand total.

Inserting a Summary

1 To insert a summary (count, sum, average, standard deviation, variance, minimum, maximum, etc.), right-click the field and choose the Insert Summary command from the shortcut menu. The Insert Summary dialog box appears.

2 Select the field you want to trigger a new summary whenever its value changes.

3 Specify the sort direction: Ascending (A to Z, 1 to 9) or Descending (Z to A, 9 to 1).

4 Crystal Reports creates a Group Header and Group Footer section and automatically places the summary in the Group Footer section.

Selecting records

When you select a field to appear in your report, Crystal Reports, by default, prints field values from every record in the active database table(s). In many cases, you may not want to include all the values, but only a subset of those values. For example, you may want to include records only for a specific group of customers or for a specific range of account numbers out of the total number of records in the active database table. With Crystal Reports, this is easy.

When you select records, you are telling Crystal Reports to base your report only on those records that meet some conditions that you set. You base those conditions on what kind of information you want in your finished report.

Assume, for example that you want a report that only shows California data. Your challenge is to find the best way to identify those records that come from California.

- If the table that you are using for your report has a state field or a region field, you can specify in your request that the program use only those records where the value in that field is equal to California.
- If the table does not have a state field and you still want to report only on California data, you may be able to identify the data in some other way. For example, if the table has a postal code or area code, you could base your record selection on the range of postal codes that apply to California (Postal Code is between *n* and *N*), or on California area codes (Area Code is one of x, y,....z).

Crystal Reports Select Records Expert makes it easy to specify the records you want included in your report. You simply select the field that you want to base your selection on and then set the selection criteria. If you want to set additional criteria for the selected field or if you want to base record selection on additional fields, the Expert provides the tools you need to do it.

You can begin your work in the Select Records Expert in one of two ways:

- Highlight the field in your report on which you want to base record selection and click the Select Records button on the button bar. The program opens the Select Records Expert, ready for you to set the conditions on the highlighted field.
- Without highlighting a report field, click the Select Records button. The Choose Field dialog box appears. Select the field you want to base your selection on and click OK. The Select Records Expert appears.

To select records

1. Open the Select Records Expert.
2. Use the drop down boxes to enter your selection criteria for the selected field.
3. If you want to base your record selection on more than one field, click the New Tab and choose your next field from the Choose Field dialog box when it appears.
4. Click OK when finished to return to your report. Crystal Reports will perform the record selection as specified.

Top N

Many times, you might want to show only the top or bottom groups in a report: the fastest selling product lines, the least productive sales regions, the states that generate the most orders. Because this kind of group selection is so popular, the program includes a facility for setting it up easily.

You set up Top N (or Bottom N) group selection using the Top N/Sort Group Expert command on the Report menu. Using this expert, you specify whether you want to display the Top N or Bottom N groups, and then you specify what number N is.

For example:

- if you want to report on the three fastest selling product lines, select top N and set N to be equal to three, or
- if you want to report on the five least productive sales regions, select bottom N and set N to be equal to five.

The program will display those groups to your specifications.

But there is one other consideration with Top N group selection and that is what to do with all the records from other groups that do not fit the Top N or Bottom N criteria you set. You need to decide whether to eliminate those records from your report entirely or to lump them altogether in a single group with the name you specify. The program enables you to do either by toggling the *include Others, with the name* check box On or Off accordingly in the Top N/Sort Group Expert.

Creating formulas

If you want to create a formula to calculate numeric values, compare one value to another and select alternative actions based on the comparison, join multiple text strings into a single string, make data calculations or comparisons, etc.:

1 Click the Insert Formula button on the button bar. The Insert Formula dialog box appears.

2 Enter a name for your formula and click OK. The Formula Editor appears. Use the Formula Editor to create, test and modify your formula.

3 Enter fields, operators, and functions into the Formula text box by double-clicking them in their respective list boxes. You can get complete information on each available Function and Operator via the Help button.

Note Each formula is a text string that can be typed directly in to the Formula text box as well as selecting components from the list boxes.

4 When finished editing, click the Accept button. Crystal Reports checks the formula syntax, and if correct, closes the Formula Editor.

5 Place the formula field where you want it to appear in your report.

Adding a graph/chart

Crystal Reports enables you to include sophisticated, colorful charts and graphs in your reports.

When you add a graph to your report, you are graphing summary and subtotal information. Before you can add a graph, therefore, you must have at least one group and one summary or subtotal in your report. For example, if you have a sales report grouped by State and a subtotal of Last Year's Sales for each State, you can quickly create a graph that will display sales by state.

Adding and modifying your graphs is easy in Crystal Reports with the use of the Graph/Chart Expert. Use the Expert every time you create or modify graphs. The Expert walks you through the creation of your graphs step-by-step.

- Click the Insert Chart button on the button bar to access the Graph/Chart Expert.
  - **Types Tab**
    When you begin creating your graphs, you will need to choose one of the many graph types available. To select a graph type, click the button that corresponds to the graph type you want.
  - **Data Tab**
    In the Data Tab, select what data you want to graph on, how often you want it to print (dependent on where you want it displayed), and what you want to show in the graph.

    The *graph on* drop-down box displays all the summaries and subtotals on your report. Simply select which one you want to base your graph on.
  - **Text Tab**
    The Text Tab allows you to enter labels for your graph by inserting titles, footnotes, and axis titles when appropriate. Simply type the text into the corresponding boxes. The program will only use the labels appropriate for the graph type that you chose earlier. The others will be ignored.
  - **Options Tab**
    The Options Tab enables you to place a legend on your graph, a maximum or minimum limit on the graph values, or a number of other options.

OLE Objects

Crystal Reports is an OLE container application. Thus it allows you to place OLE objects in your report. To do this:

1. Choose the Object command from the Insert menu.
2. Insert an existing object by choosing the *Create from file* option and selecting the desired object, or create a new object by choosing the *Create New* option then an object type. The corresponding application will open where you can create the object as desired.
3. If you want to edit the object, simply double-click it and the program in which it was created (or a similar application that allows such editing) will open.
4. Modify the object as desired, save it and Crystal Reports automatically updates the object in your report.

Adding a report title

1. To add a title to your report, click the Insert Text Field button on the button bar. The Edit Text Field dialog box appears.
2. Type the desired title into the Enter Text edit box and click Accept when finished.
3. Place the text field where you want it to appear in the Title section of your report.

Previewing your report

1 To preview your report at any time, click the Print Preview button on the button bar. The Preview Tab appears displaying the results of your report.

2 You can fine tune your report in the Preview Tab if you wish or close the window and continue working on your report in the Report Designer. The functionality is the same in both places.

Saving your report

- Save your report often by clicking the Save button on the button bar.

Printing your report

1 If you want to print a hard copy of your report, click the Printer button on the button bar. The Print dialog box appears.

2 Select the pages you want to print, the number of copies you want, etc., as desired.

3 Click OK. Crystal Reports prints your report.

That's it. It's that easy to build a report with Crystal Reports.

CHAPTER 3

Formulas

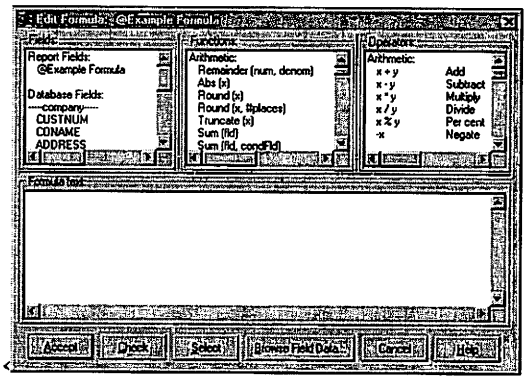
What are Formulas?

In many cases, the data you want to appear on your report already exists in fields in database tables. To prepare an order list, for example, you simply need to place the appropriate fields on your report.

Sometimes, however, you need to put data on your report that does not exist in any of the data fields. In such cases, you need to create a formula.

To create formulas, you simply click the **Insert Formula** button on the button bar. The **Formula Editor** appears.

Figure 3.1 Crystal Reports Formula Editor



Use the **Formula Editor** to create, test and modify your formula.

Note When you double-click items from the list boxes, Crystal Reports enters them in your formula complete with the brackets, punctuation, and other syntax items the **Formula Editor** needs to process them correctly. If you enter your formula manually, using the keyboard, you have to make certain you enter those syntax items yourself. It is generally safer and quicker to build a formula by choosing list box items.

Note The list of records in the **Browse Field Data** dialog box represents only a subset of the available records.

When you are finished entering your formula, click the **Accept** button. Crystal Reports checks the formula syntax, closes the **Formula Editor** if the syntax is correct, and returns you to your report. Place the formula where you want it to appear in your report.

Some of the other typical uses for formulas are:

- creating calculated fields to add to your report,

  ```
  {orders detail.UNIT PRICE}*.85
  ```

  «Calculates a price discounted 15%.»

- formatting text on a report,

  ```
  UpperCase ({customer.CUSTOMER NAME})
  ```

  «Changes all the values in the Customer Name field to upper case. Search for *Uppercase function* in Crystal Reports online Help.»

- pulling out a portion, or portions, of a text string,

  ```
  {customer.CUSTOMER NAME} [1]
  ```

  «Extracts the first letter of the customer name. Search for *Subscript Operator* in Crystal Reports online Help.»

- pulling out a portion of a date,

  ```
  Month ({orders.ORDER DATE})
  ```

  «Determines what month an order was placed in.»

These examples just scratch the surface. If you have a need for specialized data manipulation, chances are you can do it with a Crystal Reports formula.

Related Topics

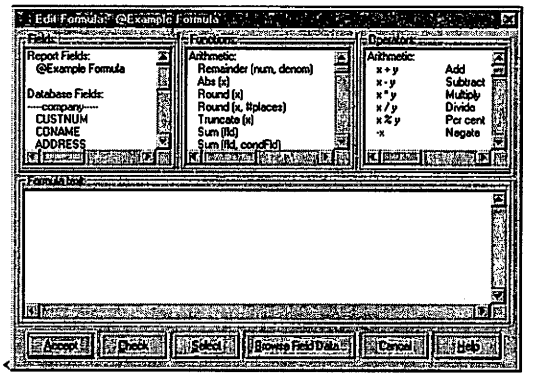
Search for *Count function* in Crystal Reports online Help.

Search for *Distinct Count function* in Crystal Reports online Help.

How formulas are created - An Introduction to the Formula Editor

The **Formula Editor** is a dialog box that contains all the tools you need to create and check your formulas.

Figure 3.2 Crystal Reports Formula Editor



Using the **Formula Editor** you combine elements to create a working formula. The three list boxes at the top of the **Formula Editor** list the primary formula components.

Fields box

The **Fields** box lists all the fields you can use as formula arguments.

You can insert fields in your formulas in two ways:

- by placing the I-beam cursor where you want the field to appear in the **Formula** text box, setting the insertion point and then double-clicking the desired field in the **Fields** box, or
- by placing the I-beam cursor where you want the field to appear in the **Formula** text box and then typing it in manually.

Note Be sure to review the correct syntax for using fields in your formulas. If you do not include the file name, leave out the separating period, or fail to surround the field in braces, Crystal Reports will generate a Formula Compiler Error message detailing your error. Search for *Syntax, checking in formulas* in Crystal Reports online Help.

Functions box

The **Functions** box lists the dozens of functions that are included with Crystal Reports. These functions are pre-built procedures that return values. All functions available are listed in this box with their arguments and arranged by their use.

You can insert functions in your formulas in the same manner as fields:

- via the **Functions** box, or
- by typing them in manually.

Search for *Fields, in formulas* in Crystal Reports online Help.

Note Be sure to review the correct syntax for using functions in your formula. If a function has required arguments, all arguments must also be entered. If any required arguments are not entered, Crystal Reports will generate a Formula Compiler Error message detailing the error. Search for *Syntax, checking in formulas* in Crystal Reports online Help.

Operators box

The **Operators** box lists the "action verbs" you can use in your formulas.

You can insert operators in your formulas in the same manner as fields and functions:

- via the **Operators** box, or
- by typing them in manually.

Search for *Fields, in formulas* in Crystal Reports online Help.

Text box

The **Formula** text box is where you actually enter the formula.

You can:

- enter the formula manually, using the keyboard,
- choose your formula components from the **Fields**, **Functions**, and **Operators** boxes, or,
- combine the two methods, choosing some of the components from the list boxes and entering other parts of the formula manually.

To insert text and numbers in formulas

You insert text and numbers in formulas by typing them directly into the **Formula** text box. You can also click the **Browse Field Data** button, highlight the value you want to insert from the list that appears, and click **Paste** to paste the value into your formula.

Note Be sure to review the correct syntax for text and numbers in formulas. Search for *Syntax, checking in formulas* in Crystal Reports online Help.

Other formula conventions

The following is a description of the remaining elements available for use in assembling a formula.

Other Formulas

Just as you can enter fields in formulas, you can enter other formulas in formulas. Crystal Reports performs the calculations in the inserted formula, and then uses the value returned by the referenced formula in the same way it uses any other value.

For example, the formula:

```
1*(2+4*6/3-7*12-8) +2*(2+4*6/3-7*12-8) +3*(2+4*6/3-7*12-8) +4*(2+4*6/3-7*12-8)= -820
```

includes the expression (2+4*6/3-7*12-8) repeated many times.

If you create a formula for the repeated expression (@F = (2+4*6/3-7*12-8)) and then reference that formula instead of entering the expression itself, you will get the same result.

```
1*{@F}+2*{@F}+3*{@F}+4*{@F} = -820
```

Every time Crystal Reports sees the formula @F, it performs the @F calculation and returns the value -82, just as the expression underlying the formula (2+4*6/3-7*12-8) returns the value -82. Search for *Order of Precedence* in Crystal Reports online Help.

To insert other formulas in formulas

You can insert other formulas into your formulas in the same manner as you did other fields:

- via the **Fields** box, or
- by typing them in manually.

Search for *Fields, in formulas* in Crystal Reports online Help.

Note Be sure to review the correct syntax for using formula fields in your formula. Search for *Syntax, checking in formulas* in Crystal Reports online Help.

Note A formula cannot reference itself.

Group Field values

Group field values are values that summarize a group (a group subtotal, a group average, and so forth). You can use them in formulas for many reasons. A typical reason would be to find out the percentage of the grand total that each group contributes (for example, what percentage of the $2,300,000 US sales figure did the Western Region contribute?).

To insert group field values in formulas

You can insert group field values into your formulas in the same manner as you did other fields:

- via the **Fields** box, or
- by typing them in manually.

Note Due to the syntax complexity of some group fields, it is highly recommended that you enter group fields by selecting them from the **Fields** box versus manually typing it in. Search for *Syntax, checking in formulas* in Crystal Reports online Help.

Formula Comments

Formula comments are notes that you include with a formula to explain its design and operation.

Comments do not print and they do not affect the formula, but they appear in the **Formula Editor**. It is always a good idea to include comments with complex formulas, especially those formulas that will be used again and again over time.

To insert comments in formulas

Type your comments in the **Formula** text box in the **Formula Editor**. The comment can be above or below the formula, or it can even follow the formula on the same line.

Note A comment must be preceded by two slashes (//).

Any of the following placements are acceptable:

```
//This is an acceptable
//position for a comment.
//Note that when we force
//the line break, we have to
//begin each new line with
//double slashes.
If {orders.ORDER AMOUNT} in (100.00 to 250.00)
Then
   .10 * {orders.ORDER AMOUNT}
Else
   0
//This is also an acceptable comment position.
If {orders.ORDER AMOUNT} > 10.00 Then
   "" //This position is also acceptable.
Else
   "Flag"
```

Formula comment considerations

The following are considerations when including comments with formulas:

- The proper syntax for a comment is two forward slashes // followed by the comment. When Crystal Reports sees the two slashes, it realizes that the text that follows for the rest of the line is comment only and not to be evaluated as part of the formula itself.
- Crystal Reports treats everything that follows the slashes on the same line as a comment.
- If you break your comment into two or more lines using the ENTER key, you must begin each new line with two slashes. If you do not, Crystal Reports treats each unslashed line as part of the formula itself and displays an error message when you check the formula syntax.

Formula Syntax

Crystal Reports requires you to enter the various components of a formula according to a specific set of rules called syntax. Syntax, like the grammar of any language, takes practice to learn and perfect. Formula components must be written in a specific way and entered in specific order. The program uses syntax items (quotation marks, brackets, parentheses, etc.) to identify the various formula components, so it is very important that you stick to the rules in order for the program to recognize your formulas as well as have them work as you had planned.

The various components of formulas, and their syntax, are listed below:

Text

```
"Text" or 'Text'
```

(enclosed within 'single' or "double quotes")

When using text in formulas, it must be enclosed within 'single' or "double" quotes. Whatever text is within those quotes will be printed.

Numbers

```
23134.7
```

When using numbers in formulas, they must be entered without any comma separators or currency symbols.

Fields

```
{file.FIELD}
```

When using fields in formulas, they must be enclosed within French braces { } with the table name to identify which database table you are referring to followed by a period and then the field name within that table.

Example

```
{customer.REGION}
```

This identifies the Region field from the Customer table.

Operators

```
1+1
```

When using operators in formulas, you simply type in the operator where desired. You may place a space before and after the operator if you desire, but to do so is optional (i.e., 1 + 1).

Functions

```
FunctionName(x)
```

When using functions in formulas, you simply type the function and supply the arguments as specified in the parentheses. For example, the Sum(x) function requires a field or formula as its (x) argument.

Example

```
Average({order.AMOUNT})
```

This formula will calculate the average of all values in the Amount field.

Brackets in formulas

Crystal Reports uses three different types of brackets in writing formulas. Each one has a specific purpose and can be used only with certain formula elements.

- { } are called French braces and are placed around database and formula fields:

  ```
  {customer.REGION}, {@sum}
  ```

- [] are called Square brackets and are placed when using the Subscript or Array Operator:

  ```
  {customer.CUSTOMER NAME} [1]
  ```

- () are called Parentheses and are placed around the arguments of a function:

  ```
  Round(x, # places), Abs(x)
  ```

Note Parentheses can also be used to control the order in which the formula elements are evaluated. Search for *Order of precedence* in Crystal Reports online Help.

Tip A useful way to remember which brackets are used where is the following:

- French {} = Fields
- Square [] = Subscript (and Array)
- Parentheses () = Parameters

Whether you enter the formula manually or by double-clicking formula components from the component list boxes, you must use the correct syntax if you want your formula to work.

How formulas are evaluated - Order of precedence

If you are creating formulas that contain different kinds of operators, it is important to consider the order in which Crystal Reports evaluates the separate parts of your formula. This order is called order of precedence.

Simple order of precedence follows basic math rules of precedence. Multiplication and division are performed first, from left to right, then addition and subtraction are performed. For example:

```
5 + 10 * 3 = 35
```

The multiplication 10 * 3 is performed first to get 30. Then, 30 is added to 5 to arrive at the final answer, 35.

Now, if your intention is to add 5 to 10 and then multiply the sum by 3, you have to modify the order of precedence with parentheses. You can do that as follows:

```
(5 + 10) * 3 = 45
```

It is clear that parentheses have a higher precedence than the add, subtract, multiply, and divide operators. They redirect the order of calculation.

Note When a formula contains other formulas such as:

```
@Extension * 107.5%,
```

the program first evaluates the embedded formula, @Extension, following order of precedence rules and then evaluates the rest of the primary formula.

How to delete formulas from your report

When you create a formula and enter it in your report, Crystal Reports:

- stores the specification for creating the formula, using the name you assigned it, and
- places a working copy of that formula at the point you specify in the report. A working copy is any occurrence of the formula in the report.

In order to delete formulas, you must delete the specification and all working copies of the formula.

Note You cannot delete the specification without deleting all working copies of the formula.

To delete individual working copies of a formula

1 Select the formula copy you want to delete in your report.

2 Press the DELETE or DEL key on your computer's keyboard.

Note Even after you have deleted all of the working copies of a formula from your report, the formula specification remains unchanged. The specification is listed in the Formula Tab of the **Insert Fields** dialog box, and it is available for immediate use should you wish to enter the formula in your report again.

Search for *fields, inserting* in Crystal Reports online Help.

To delete the specification

Note This step can only be completed after you have deleted all working copies of the formula from your report.

1 Once you have deleted all working copies of the formula, click the **Insert Fields** button on the toolbar. The **Insert Fields** dialog box appears with the Database Tab active.

2 Click the Formula Tab to activate it.

3 Select the formula specification you want to delete from the Formula name list. Crystal Reports activates the **Delete** button.

4 Click the **Delete** button, and Crystal Reports deletes the formula specification.

Note If the formula is being used in another formula the program will delete the specification nonetheless.

Note If you have not deleted all working copies of the selected formula, the program displays the message: **Formula field in use, cannot be deleted!**

How to copy formulas from Crystal Reports online Help

Windows allows you to copy text from online Help topics to the Clipboard. You can then paste this text wherever it is needed. Since the formulas you develop using the **Formula Editor** are simply text, you can save yourself a lot of time by copying useful formulas directly into the **Formula Editor** and then modifying them to fit your needs.

To copy formulas from online Help

1 With the **Formula Editor** active, call up the Crystal Reports online Help in any of the standard ways (**Help** menu, F1 function key, etc.).

2 Regardless of the topic that first appears, use the **Search** or **Find** facility to find the topic that contains the formula of interest.

3 Call up the topic.

4 Choose the **Copy** command from the **Edit** menu. The **Copy** dialog box appears with the topic text displayed in a scrolling edit window.

Note To do this you highlight the text and choose **Copy** from the **Edit** menu to copy the text to the Clipboard.

5 Scroll through the topic until you locate the formula you want to copy.

6 Select the formula by dragging the I-beam cursor over it, and click the **Copy** button when finished. Windows puts a copy of the selected text on the Clipboard.

7 Place the insertion point where you want the text to appear in the **Formula Editor** and press CTRL-V to paste the text from the Clipboard.

8 Modify the formula to fit your needs.

Note In Windows 95 ad NT, highlight the text you want to copy and choose **Copy** from the **Edit** menu or press CTRL-C to copy the text to the Clipboard.

Note Change the fields, formulas, group fields, conditional statements, and text strings as necessary for use with the data used in the new report.

CHAPTER 4

Functions

Introduction

This chapter provides an in-depth look at all Crystal Reports functions.

Functions are built-in procedures or subroutines used to evaluate, make calculations on, or transform data. When you specify a function, Crystal Reports performs the set of operations built into the function without you having to specify each operation separately. In this way, a function is a kind of shorthand that makes it easier and less time consuming for you to create formulas. Simple, straightforward examples are provided for each function. In addition, for many functions, references are provided to formulas in the Formulas in Action topic in Crystal Reports online Help (search for Formulas in Action). Formulas in Action shows advanced uses of the functions to solve complex, real-world reporting problems.

Abs (x)

Arguments

x is any number.

Returns

Absolute value of x.

Action

Abs returns the absolute value of x, eliminating any negative value.

Aged0To30Days
Aged31To60Days
Aged61To90Days

Returns

None

Action

Specifies a range of date values that fall within a certain time period before the present date. If the current date is 12/30/98, Aged0To30Days specifies the period from 12/1/98 to the present date, Aged31To60Days specifies the period from 11/1/98 to 11/30/98, and Aged61To90Days specifies the period 10/2/98 to 10/31/98.

AllDatesFromToday

Returns

None

Action

Specifies a range of date values that includes any date from the present day to any future date value that may appear in a field. AllDatesFromToday includes the present day.

AllDatesFromTomorrow

Returns

None

Action

Specifies a range of date values that fall after the present day. AllDatesFromTomorrow does not include the present day, but does include any future date.

AllDatesToToday

Returns

None

Action

Specifies a range of date values that includes every day up through the present day. AllDatesToToday includes the present day.

AllDatesToYesterday

Returns

None

Action

Specifies a range of date values that includes every day up through the previous day. AllDatesToYesterday includes all dates before today, but does not include the present day.

Average (fld)
Average (fld, condFld)
Average (fld, condFld, cond)
Average (x)

Action

Crystal Reports enables you to average the values that appear in your report. For example:

- If a sales report includes a field that shows the amount of each order, you can compute the average of all the orders that appear on the report (a grand total average). See *Summary Function (fld),* later in this chapter.
- If you break orders into groups (for example, orders grouped by the state that they come from), you can compute the average order per group (in this case, per state). See *Summary Function (fld, condFld)*, later in this chapter.

- If you break orders into date or Boolean groups (for example, orders grouped by the month in which they were placed), you can compute the average order per group based on a particular change in the date or Boolean field (in this case, per month). See *Summary Function (fld, condFld, cond),* later in this chapter.
- If you specify an array of individual values, you can also compute the average value in the set. See *Summary Function (x),* later in this chapter.

BeforeReadingRecords

Returns

None

Action

Specifies that the formula is to be evaluated before the database records are read.

Calendar1stHalf
Calendar2ndHalf

Returns

None

Action

Specifies a range of date values that includes all dates that fall within the first or second half of the calendar year, respectively. The first half of the calendar year includes all dates from January 1st through June 30th. The second half of the calendar year includes all dates from July 1st through December 31st.

Calendar1stQtr
Calendar2ndQtr,
Calendar3rdQtr
Calendar4thQtr

Returns

None

Action

Specifies a range of date values that falls within the 1st, 2nd, 3rd, or 4th quarter of the calendar year. The first quarter of the calendar year includes all dates from January 1st through March 31st. The second quarter of the calendar year includes all dates from April 1st through June 30th. The third quarter of the calendar year includes all dates from July 1st through September 30th. The fourth quarter of the calendar year includes all dates from October 1st through December 31st.

Count (fld)
Count (fld, condFld)
Count (fld, condFld, cond)
Count (x)

Action

Crystal Reports enables you to count the values that appear in your report. For example:

- If a sales report includes all orders made and the amount of each order, you can compute the total number of orders that appear on the report (a grand total count). See *Summary Function (fld),* later in this chapter.
- If you break orders into groups (for example, orders grouped by the state that they come from), you can compute the number of orders per group (in this case, per state). See *Summary Function (fld, condFld),* later in this chapter.
- If you break orders into date or Boolean groups (for example, orders grouped by the month in which they were placed), you can compute the number of orders per group based on a particular change in the date or Boolean field (in this case, per month). See *Summary Function (fld, condFld, cond),* later in this chapter.
- If you specify a set of individual values, you can compute the number of values in the set. See *Summary Function (x),* later in this chapter.

Date (year, month, day)

Arguments

- *year* is a whole number representing a year, e.g., 1996.
- *month* is a whole number representing a month, e.g., 12 for December.
- *day* is a whole number representing a day of the month, e.g., 05.

Returns

Date Value

Action

Date returns a date value based on the individual elements of a date provided: year, month, and day. If the individual elements of the date are provided:

- The year must be four digits.
- The month must be a number from 1 to 12.
- The day must be a number from 1 to 31.

Day (x)

Arguments

x is a date value or a date/time value.

Returns

Whole Number (the day of the month)

Action

Day extracts the day from a date or date/time value and returns a whole number.

DayOfWeek (x)

Arguments

x is a date value or dateTime value.

Returns

Whole Number

Action

DayOfWeek extracts the day component of a date, determines the day of the week the date falls on, and converts the day of the week to a number (1 to 7) where Sunday is the first day of the week.

DistinctCount (fld)
DistinctCount (fld, condFld)
DistinctCount (fld, condFld, cond)
DistinctCount (x)

Action

Crystal Reports enables you to get a distinct count of the values that appear in your report. For example:

- If a sales report includes all orders made by customers, you can compute the total number of distinct customers that appear in the report (a grand total distinct count), excluding any duplicate records. If a customer made more than one order, the duplicate occurrences of that customer are ignored. See *Summary Function (fld)*, later in this chapter.
- If you break orders into groups (for example, orders grouped by the state that they come from), you can compute the number of distinct customers per group (in this case, per state). Any customers that made more than one order and appear more than once in a group are only counted once. See *Summary Function (fld, condFld)*, later in this chapter.
- If you break orders into date or Boolean groups (for example, orders grouped by the month in which they were placed), you can compute the number of distinct customers in each group based on a particular change in the date or Boolean field (in this case, the number of customers that placed orders each month). If a customer placed more than one order within the month, duplicate instances of that customer are ignored. See *Summary Function (fld, condFld, cond)*, later in this chapter.
- If you specify a set of individual values, you can compute the number of distinct values in the set. Duplicate values in the set are only counted once. See *Summary Function (x)*, later in this chapter.

GroupNumber

Returns

Whole Number

Action

Returns the current group number.

IsNull (fld)

Arguments

fld is any valid database field.

Returns

Boolean Value

Action

Evaluates the field specified in the current record and returns TRUE if the field contains a null value.

Last4WeeksToSun

Returns

None

Action

Specifies a range of dates that includes the four weeks previous to last Sunday. A week begins on a Monday and ends on a Sunday. For example, September, 1996, begins on a Sunday.

- If today is September 22 (Sunday), Last4WeeksToSun begins on August 26 (Monday) and ends on September 22 (today, Sunday).
- If today is September 28 (Saturday), Last4WeeksToSun begins on August 26 (Monday) and ends on September 22 (the previous Sunday).

Last7Days

Returns

None

Action

Specifies a range of date values that includes all dates from seven days ago to today (including today).

LastFullMonth

Returns

None

Action

Specifies a range of date values that includes all dates from the first to last day of the previous month.

LastFullWeek

Returns

None

Action

Specifies a range of date values that includes all dates from Sunday to Saturday of the previous week.

LastYearMTD

Returns

None

Action

Specifies a range of date values in the previous year that matches the current month to date.

LastYearYTD

Returns

None

Action

Specifies a range of date values that includes all dates in the last year, up to the current date last year.

Length(str)

Arguments

str is a text string or a field containing a text string.

Returns

Whole Number

Action

Length returns the number of characters in the specified text string.

Note Crystal Reports includes any blank spaces as part of the character count.

Note Text strings must be enclosed in either double ("") or single (' ') quotation marks.

LowerCase(x)

Arguments

x is a text string to be converted to all lower case characters.

Returns

Text String

Action

LowerCase prints the text string or text string value in the data field in all lower case letters.

Maximum (fld)
Maximum (fld, condFld)
Maximum (fld, condFld, cond)
Maximum (x)

Action

Crystal Reports enables you to find the maximum value that appears in a set of values. For example:

- If a sales report includes a field that shows the amount of each order, you can find the highest order amount of all the orders that appear on the report (a grand total maximum amount). See *Summary Function (fld)*, later in this chapter.
- If you break orders into groups (for example, orders grouped by the state that they come from), you can find the highest order amount per group (in this case, per state). See *Summary Function (fld, condFld)*, later in this chapter. .
- If you break orders into date or Boolean groups (for example, orders grouped by the month in which they were placed), you can find the highest order amount per group based on a specific change in the date or Boolean field (in this case, per month). See *Summary Function (fld, condFld, cond)*, later in this chapter.
- If you specify a set of individual values, you can find the highest value in the set. See *Summary Function (x)*, later in this chapter.

Minimum (fld)
Minimum (fld, condFld)
Minimum (fld, condFld, cond)
Minimum (x)

Action

Crystal Reports enables you to find the minimum value that appears in a set of values. For example:

- If a sales report includes a field that shows the amount of each order, you can find the lowest order amount of all the orders that appear on the report (a grand total minimum amount). See *Summary Function (fld)*, later in this chapter.
- If you break orders into groups (for example, orders grouped by the state that they come from), you can find the lowest order amount per group (in this case, per state). See *Summary Function (fld, condFld)*, later in this chapter.

- If you break orders into date or Boolean groups (for example, orders grouped by the month in which they were placed), you can find the lowest order amount per group based on a specific change in the date or Boolean field (in this case, per month). See *Summary Function (fld, condFld, cond)*, later in this chapter.
- If you specify a set of individual values, you can find the lowest value in the set. See *Summary Function (x)*, later in this chapter.

Month(x)

Arguments

x is a date value or a dateTime value.

Returns

Whole Number

Action

Month extracts the month component of a date or dateTime value and converts it to a number.

MonthToDate

Returns

None

Action

Specifies a range of date values that includes all dates from the first day of the month to today.

Next(fld)

Arguments

fld is any valid database or formula field.

Returns

A field value of the same type as the argument.

Action

Next(fld) returns the value of the specified field for the next record.

Next30Days
Next31To60Days
Next61To90Days
Next91To365Days

Returns

None

Action

Specifies a range of date values that includes all dates in the period specified starting from today (including today).

NextIsNull(fld)

Arguments

fld is any valid database or formula field.

Returns

Boolean Value

Action

Evaluates the field specified in the next record and returns a TRUE if the field contains a null value.

NumericText(str)

Arguments

str is a text string being tested for numeric text.

Returns

Boolean Value

Action

NumericText tests to see if the content of a text object is a number.

- If the entire content of the field is a number or if the characters extracted via the subscript operator are entirely a number the expression returns a TRUE value.
- If any part of the content of the field or of the characters extracted are not a number, the expression returns a FALSE value.

Over90Days

Returns

None

Action

Specifies a range of date values that includes all values that are more than 90 days older than the current date.

PageNumber

Returns

Whole Number

Action

PageNumber inserts the current page number as a field in a formula.

PopulationStdDev (fld)
PopulationStdDev (fld, condFld)
PopulationStdDev (fld, condFld, cond),
PopulationStdDev (x)

Action

Crystal Reports enables you to find the population standard deviation of a set of values in your report. For example:

- You can calculate the grand total population standard deviation for all values in a field. See *Summary Function (fld)*, later in this chapter.
- You can calculate the population standard deviation for all values within a group (for example, sales grouped by the state that they come from). See *Summary Function (fld, condFld)*, later in this chapter.
- You can calculate the population standard deviation for all values within a group in which grouping is controlled by changes in a date or Boolean field (for example, sales grouped by the month in which they were made). See *Summary Function (fld, condFld, cond)*, later in this chapter.
- If you specify a set of individual values, you can compute the population standard deviation of the values in the set. See *Summary Function (x)*, later in this chapter.

PopulationVariance (fld)
PopulationVariance (fld, condFld)
PopulationVariance (fld, condFld, cond)
PopulationVariance (x)

Action

Crystal Reports enables you to find the population variance of a set of values in your report. For example:

- You can calculate the grand total population variance for all values in a field. See *Summary Function (fld)*, later in this chapter.
- You can calculate the population variance for all values within a group (for example, sales grouped by the state that they come from). See *Summary Function (fld, condFld)*, later in this chapter.
- You can calculate the population variance for all values within a group in which grouping is controlled by changes in a date or Boolean field (for example, sales grouped by the month in which they were made). See *Summary Function (fld, condFld, cond)*, later in this chapter.
- If you specify a set of individual values, you can compute the population variance of the values in the set. See *Summary Function (x)*, later in this chapter.

Previous(fld)

Arguments

fld is any valid database or formula field.

Returns

A field value of the same type as that passed as the fld argument.

Action

Previous(fld) returns the value of the specified field in the previous record.

PreviousIsNull(fld)

Arguments

fld is any valid database or formula field.

Returns

Boolean Value

Action

Evaluates the field specified in the previous record and returns a TRUE if the field contains a null value.

PrintDate

Returns

Date Value

Action

PrintDate inserts the date the report is printed as a field in a formula.

RecordNumber

Returns

Whole Number

Action

RecordNumber returns the current record number.

Remainder (num, denom)

Arguments

- *num* is a fractional value.
- *denom* is a fractional value.

Returns

Fractional Value

Action

Remainder returns the remainder after the numerator (dividend) has been divided by the denominator (divisor). In a typical division situation, Crystal Reports expresses a quotient as a whole number (if any) and up to six decimal places. When using Remainder, however, Crystal Reports performs the division internally, determines the whole number quotient and the remainder, and returns only the remainder.

ReplicateString (str, #copies)

Arguments

- *str* is the text string to be replicated.
- *#copies* is a whole number indicating the number of times str is to be replicated.

Returns

Text String

Action

Replicates the string in str the number of times specified by #copies.

Round(x)
Round(x, #places)

Arguments

- *x* is a fractional value to be rounded off.
- *#places* is a whole number indicating the number of decimal places x is to be rounded off to.

Returns

Fractional Number

Action

Round rounds to the nearest whole number if the #places argument is excluded. If the value to the right of the decimal point is .499 or below, Crystal Reports rounds to the next lowest number. If the value to the right of the decimal point is .5 or above, Crystal Reports rounds to the next highest number.

If the #places argument is used, the value in x is rounded to the nearest decimal place indicated by #places. Specifying #places as 0 works identical to leaving the #places argument off. If #places is negative, the number is rounded to the nearest tens, hundreds, and so on.

StdDev (fld)
StdDev (fld, condFld)
StdDev (fld, condFld, cond)
StdDev (x)

Action

Crystal Reports enables you to find the standard deviation of a set of values in your report. For example:

- You can calculate the grand total standard deviation for all values in a field. See *Summary Function (fld),* later in this chapter.
- You can calculate the standard deviation for all values within a group (for example, sales grouped by the state that they come from). See *Summary Function (fld, condFld),* later in this chapter.
- You can calculate the standard deviation for all values within a group in which grouping is controlled by changes in a date or Boolean field (for example, sales grouped by the month in which they were made). See *Summary Function (fld, condFld, cond),* later in this chapter.
- If you specify a set of individual values, you can compute the standard deviation of the values in the set. See *Summary Function (x),* later in this chapter.

Sum (fld)
Sum (fld, condFld)
Sum (fld, condFld, cond)
Sum (x)

Action

Crystal Reports enables you to add the values that appear in your report. For example:

- If a sales report includes a field that shows the amount of each order, you can compute the sum of all the orders that appear on the report (a grand total sum). See *Summary Function (fld),* later in this chapter.
- If you break orders into groups (for example, orders grouped by the state that they come from), you can compute the sum of the order amounts per group (in this case, per state). See *Summary Function (fld, condFld),* later in this chapter.

- If you break orders into date or Boolean groups (for example, orders grouped by the month in which they were placed), you can compute the sum of the order amounts per group based on a specific change in the date or Boolean field (in this case, per month). See *Summary Function (fld, condFld, cond),* later in this chapter.
- If you specify a set of individual values, you can also compute the sum of the values in the set. See *Summary Function (x),* later in this chapter.

Summary Function (fld)

Arguments

fld is any valid database or formula field that can be evaluated by the function.

Returns

Fractional Number

Action

Summary Function(fld) summarizes the values in the specified field for the entire report. It generates, in effect, a "grand total" summary.

Summary Function (fld, condFld)

Arguments

- *fld* is any valid database or formula field that can be evaluated by the function.
- *condFld* is any valid database or formula field used to group the values in fld by.

Returns

Fractional Number

Action

Summary Function(fld, condFld) summarizes each group of values that is generated when the specified summary condition is met.

Summary Function (fld, condFld, cond)

Arguments

- *fld* is any valid database or formula field that can be evaluated by the function.
- *condFld* is any valid date or Boolean field used to group the values in fld by.
- *cond* is a text string indicating the condition of condFld that controls grouping. Valid strings for this argument depend on whether condFld contains date or Boolean values.

Returns

Fractional Number

Action

Summary Function(fld, condFld, cond) summarizes each group of values that is generated when the specified summary condition is met. These functions work just like Summary Function(fld, condFld), but, because they use a date or Boolean field as a sort and group by (trigger) field, they require a condition in addition to the other arguments.

Summary Function (x)

Arguments

x is an array of values that can be evaluated by the function being used.

Returns

Fractional Number

Action

Summary Function(x) summarizes the values in an array of constants, data field values, or formulas (a*b, c/d, etc.) separated by commas.

Today

Returns

Date Value

Action

Today returns the current date.

ToNumber(x)

Arguments

x is a text string that holds numeric text.

Returns

Fractional Number

Action

ToNumber converts a text string to a number.

In a database, some numbers are stored in numeric fields, as numbers, and some are stored in string fields, as text. You make the determination which fields are to be numeric and which are to be text when you set up the database in the first place. Numbers on which you might wish to perform arithmetic (item cost, quantity ordered, etc.) are typically stored in numeric fields; numbers on which you do not expect to perform arithmetic (customer number, telephone number, etc.) are typically stored in text objects.

ToNumber allows you to convert a number stored as text to a number on which you can perform arithmetic.

ToText (x)
ToText (x, #places)

Arguments

- *x* is a number you wish to be converted into a text string.
- *#places* is a whole number indicating the number of decimal places to carry the value in x to.

Returns

Text String

Action

The ToText function converts a number to a text string.

ToWords (x)
ToWords (x, #places)

Arguments

- *x* is a fractional number to be converted into words.
- *#places* is a whole number indicating the number of decimal places to be converted. (This argument is optional.)

Returns

Text String

Action

You can use this function to convert a number or currency field value or the result of a numeric calculation to words so it can be used as text. The ability to adjust the number of decimal places can be useful when the number is the result of a calculation that may produce more decimal places than you want.

TrimLeft (str)

Arguments

str is a text string stored right-justified.

Returns

Text String

Action

TrimLeft removes all spaces to the left of a string or data field which is stored as a right-justified string in a database.

TrimRight (str)

Arguments

str is a text string stored left-justified.

Returns

Text String

Action

TrimRight removes all spaces to the right of a string or data field which is stored left-justified in a database.

Truncate (x)

Arguments

x is a fractional number to be truncated.

Returns

Fractional Number

Action

Truncate(x) returns a whole number by truncating the number at the decimal point.

UpperCase (str)

Arguments

str is a text string.

Returns

Text String

Action

UpperCase prints the text string or text value in the data field in uppercase (capital letters).

Variance (fld)
Variance (fld, condFld)
Variance (fld, condFld, cond)
Variance (x)

Action

Crystal Reports enables you to find the variance of a set of values in your report. For example:

- You can calculate the grand total variance for all values in a field. See *Summary Function (fld),* earlier in this chapter.
- You can calculate the variance for all values within a group (for example, sales grouped by the state that they come from). See *Summary Function (fld, condFld),* earlier in this chapter.
- You can calculate the variance for all values within a group in which grouping is controlled by changes in a date or Boolean field (for example, sales grouped by the month in which they were made). See *Summary Function (fld, condFld, cond),* earlier in this chapter.
- If you specify a set of individual values, you can compute the variance of the values in the set. See *Summary Function (x),* earlier in this chapter.

WeekToDateFromSun

Returns

None

Action

Specifies a range of date values that includes all days from last Sunday to Today (including today).

WhilePrintingRecords

Returns

None

Action

Forces the program to evaluate the formula while it is printing database record data.

WhileReadingRecords

Returns

None

Action

Forces the program to evaluate the formula while it is reading database record data.

Year(x)

Arguments

x is a date value or a date/time value.

Returns

Whole Number

Action

Year extracts the year from a date and returns it as a number.

YearToDate

Returns

None

Action

Specifies a range of date values that includes all days from the first day of the calendar year to today.

CHAPTER 5

Operators and Variables

Introduction

This chapter provides an in-depth look at all Crystal Reports operators.

Operators are special symbols or words that describe an operation or an action to take place between two or more values. Operators are used in formulas. Crystal Reports reads the operators in a formula and performs the actions specified.

This chapter provides an in-depth explanation of each operator. Operators are listed in alphabetical order.

Note Crystal Reports operators (Equal, In string, etc.) are case sensitive. Thus "ABC" is not equal to "abc", "abc" is not in the string "ABCDEF", etc.

Add (+)

Usage

x + y

«Add values x and y.»

Example(s)

```
5 + 6
{file.QTY1} + {file.QTY2}
```

«Returns 1487, where Qty1 = 366 and Qty2 = 1121.»

```
{file.AMT1} + {file.AMT2} + {file.AMT3} +    {file.AMT4}
```

«Returns 20, where Amt1 = 2, Amt2 = 4, Amt3 = 6, Amt4 = 8.»

```
{file.CLASS1} + 25
```

«Returns 37, where Class1 = 12.»

```
Date(1991,04,15) + 12
```

«Returns Apr 27 91»

And (and)

Usage

x and y

- False and False = False
- False and True = False
- True and False = False
- True and True = True

Example(s)

```
If {file.CREDIT LIMIT} = 5000 And
{file.SALESMAN} = "SP" Then
   {file.AMOUNT}
Else
   0;
```

«If the credit limit is 5000 and the salesman is SP (both conditions true) then return the value in the Amount field, otherwise return zero.»

```
A > B and B > C
```

«Returns TRUE where A = 10, B = 6, and C = 3 (both conditions true).»

```
A > B and B > C
```

«Returns FALSE where A =10, B =6, and C =7 (only 1 of the 2 conditions true).»

```
(A>B) and (A*C-D>E) and (E/D<=B)
```

«Returns TRUE where A = 7, B = 5, C = 3, D = 2, E = 10 (all three of the conditions are true).»

Assignment (:=)

Usage

x := n

«Assigns the value n to the variable x. (x must have already been declared in the same formula.)»

Example(s)

```
Amount := 0
```

«Initializes (zero's out) the Amount variable.»

```
Amount := 100
```

«Assigns the value 100 to the Amount variable.»

```
Amount := Amount + {detail.QTY}
```

«Assigns the result of a calculation to the Amount variable. The calculation adds the value of the quantity field ({detail.QTY}) to the current value of the Amount variable.»

```
Amount := {detail.QTY1} + {detail.QTY2} +    {detail.QTY3}
```

«Totals the three quantity fields and assigns the total to the Amount variable.»

```
Customer := "Westside Motors"
```

«Assigns the string "Westside Motors" to the Customer variable.»

```
Customer := TrimRight({customer.FIRST NAME}) + " "  + {customer.LAST NAME}
```

«Trims the trailing blanks from the first name field {customer.FIRST NAME}, adds a space, concatenates the last name field {customer.LAST NAME}, and assigns the concatenated value of both fields to the Customer variable.»

```
Customer:= "Mr. " + {customer.LAST NAME}
```

«Concatenates the string "Mr. " with the value of the last name field {customer.LAST NAME}, and assigns the concatenated value to the Customer variable.»

```
Amount := 100; Customer := "Westside Motors"
```

«Assigns the constant 100 to the number variable Amount and assigns the string "Westside Motors" to the string variable Customer. You can assign values to multiple variables by separating the assignment statements with semicolons.»

Comment (//)

Usage

// text

«The text that follows the operator is a comment; it is not treated as part of the formula, and it does not print.»

Example(s)

```
{file.SALES} - {file.COST}
// calculates the gross profit
```

«{file.SALES} - {file.COST} is the formula; the Formula Editor ignores everything else.»

```
({file.SALES} - {file.QUOTA}) * .06
//calculates sales commission
```

«({file.SALES} - {file.QUOTA}) * .06 is the formula; the Formula Editor ignores everything else.»

```
// What follows is
// a formula. Note that when
// we force the line break we have
// to begin each new line with a
// comment operator
If {file.COST} > {file.SELLPRICE} Then
   "Loss"
Else
   "";
//That was a formula
```

«All commented lines that appear before or after the formula are ignored.»

Concatenate (+)

Usage

x + y

«Concatenates (connects) string x to string y to make one contiguous string.»

Example(s)

```
"Bread " + "and " + "butter"
```

«Returns "Bread and butter".»

```
"Your customer number is " + ({customer.CUSTOMER     ID}) + " and your company contact
person is " +  ({customer.FIRST NAME}) + "."
```

«Returns "Your customer number is 64 and your company contact person is Bob." where CustomerID = 64 and First Name = Bob.»

Divide (/)

Usage

x/y

«Divide value x by value y.»

Example(s)

```
25/5
```

«Returns 5.»

```
1/3
```

«Returns .33333.»

```
{file.SALES} / {file.FORECAST}
```

«Returns .875, where Sales = 52533, Forecast = 60000.»

```
{file.DAYSDUE} / 5
```

«Returns 22, where DaysDue = 110.»

Note If the denominator = 0, the report will be halted with a divide by zero warning. If you want to avoid this type of problem, you should put in a test similar to the following:

```
If {file.FORECAST} = 0 Then
   0
Else
   {file.SALES} / {file.FORECAST}
```

Equal (=)

Usage

x = y

«x is equal to y.»

The Equal operator tells Crystal Reports to evaluate an expression (x = y) and return a TRUE (if x is equal to y) or FALSE (if x is not equal to y).

Example(s)

```
{file.QUANTITY} = 3
```

«True, where Quantity has a value of 3.»

```
{file.QUANTITY} = 3
```

«False, in all other situations.»

```
{file.YTD} = {file.LAST YEAR YTD}
```

«True, where the value of the field YTD is identical to the value of the field Last Year YTD.»

```
{file.YTD} = {file.LAST YEAR YTD}
```

«False, in all other situations.»

```
({file.SALES} - {file.COGS}) = 22,554
```

«True, where Sales = 109,986 and COGS = 87,332.»

```
({file.SALES} - {file.COGS}) = 22,654
```

«False, in all other situations.»

```
{customer.LAST NAME} = "Johnson"
```

«True, where the text string in the Last Name field is Johnson.»

```
{customer.LAST NAME} = "Johnson"
```

«False, in all other situations.»

Greater Than (>)

Usage

x > y

«x is greater than y.»

Example(s)

```
{file.WEIGHT} > 200
```

«False, where Weight = 150, Weight = 199, or Weight = 200.»

```
{file.WEIGHT} > 200
```

«True, where Weight = 400 or Weight = 201.»

```
{file.COST} > {file.PRICE}
```

«True, where Cost = 350 and Price = 325.»

Greater Than Or Equal (>=)

Usage

x >= y

«x is greater than or equal to y.»

Example(s)

```
{file.WEIGHT} >= 200
```

«False, where Weight = 150 or Weight = 199.»

```
{file.WEIGHT} >= 200
```

«True, where Weight = 400, Weight = 200, or Weight = 201.»

```
{file.COST} >= {file.PRICE}
```

«False, where Cost = 350, Price = 400.»

If-Then-Else

Usage

If x Then y Else z

«If x is true Then do y. If x is not true (Else), do z.»

Example(s)

```
If {customer.POSTAL CODE} <= "49999" Then
   "Blue Label"
Else
   "Ground";
```

«Assigns method of shipping based on distance from ship point.»

```
If ToNumber({file.ITEM}) >= 2500 And
ToNumber({file.ITEM}) < 2600 Then
   "Seasonal"
Else
   "";
```

«If statement includes an And operator.»

```
If {file.COUNT} >= 25 Then
   {file.DISTRIBUTOR} * {file.COUNT}
Else
   {file.DEALER} * {file.COUNT};
```

«Quantity ordered determines price list used.»

```
If {file.ONHAND} > 10 Then
   {file.ORDERED}
Else
   If {file.ORDERED} < 5 Then
      {file.ORDERED}
   Else
      2;
```

«Allocation based on quantity ordered using nested If-Then-Else.»

In Array (in)

Usage

x in [y]

«Is x in the array y?»

Example(s)

```
{customer.REGION} in ["CA","HI","AK"]
```

«Is the value of the Region field in the array of state abbreviations listed in the brackets?»

```
{file.COLOR} in ["Red","White","Blue"]
```

«Is the value of the Color field in the array of colors listed in the brackets?»

```
DayofWeek({orders.ORDER DATE}) in [2,4,6]
```

«Is the value of the Order Date field, converted to a number that represents the day of the week, in the array of numbers listed in the brackets? (Sunday = 1, Saturday = 7)»

In Range (to)

Usage

x in y

«Tests a range of values (y) to see if a value (x) falls within the range specified.»

Example(s)

```
Today in Date(1990,09,01) to Date(1990,09,20)
```

«True, where Today = September 15, 1990.»

```
Today in Date(1990,09,01) to Date(1990,09,20)
```

«False, where Today = September 21, 1990.»

```
{file.QTY} in {file.ONHAND} to ({file.BACKORDER} +  {file.ONORDER})
```

«True, where Qty=20, OnHand=10, Backorder=5, OnOrder=25. (Is 20 in the range that begins with 10 and ends with the sum of 5 and 25?)»

```
{file.QTY} in {file.ONHAND} to ({file.BACKORDER} +  {file.ONORDER})
```

«False, where Qty=31, OnHand=10, Backorder=5, OnOrder=25. (Is 31 in the range that begins with 10 and ends with the sum of 5 and 25?)»

In String (in)

Usage

x in y

«Test for the presence of string x in string y.»

Example(s)

```
"Elm" in {customer.ADDRESS1}
```

«True, where Address is 1335 Elm Street.»

```
"elm" in {customer.ADDRESS1}
```

«False, where Address is 1335 Elmer Street.»

```
"elm" in {file.MOTTO}
```

«False, where Motto = "Feel more energy".»

(The "el" ending of "feel" and the "m" beginning of the word "more" are separated by a space which itself counts as an element.)

```
"el m" in {file.MOTTO}
```

«True, where Motto = "Feel more energy".»

(The search string "el m" this time contains the space between the l and the m which allows for a perfect match.)

Note The "in" operator can also be used to test for the presence of a string in a text range (i.e., "V5B" in "V0A" to "V9Z").

Less Than (<)

Usage

x < y

«x is less than y.»

The less than operator tells the Formula Editor to evaluate an expression (x < y) and return a TRUE (if x is less than y) or FALSE (if x is equal to or greater than y).

Example(s)

```
{file.WEIGHT} < 200
```

«True, where Weight = 150 or Weight = 199.»

```
{file.WEIGHT} < 200
```

«False, where Weight = 200 or Weight = 400.»

```
{file.COST} < {file.PRICE}
```

«True, where Cost = 350 and Price = 400.»

```
{file.COST} < {file.PRICE}
```

«False, where Cost=350 and Price=350, or where Cost=350 and Price=325.»

Less Than Or Equal (<=)

Usage

x <= y

«x is less than or equal to y.»

Example(s)

```
{file.WEIGHT} <= 200
```

«True, where Weight = 150, Weight = 200, or Weight = 199.»

```
{file.WEIGHT} <= 200
```

«False, where Weight = 400.»

```
{file.COST} <= {file.PRICE}
```

«True, where Cost=350, Price=400, or where Cost=350 and Price=350.»

```
{file.COST} <= {file.PRICE}
```

«False, where Cost = 350 and Price = 325.»

Like

Usage

x like y

{file.FIELD} like "c?n*"

«This operator tests to see if the contents of {file.FIELD} matches a pattern that you specify in a character string "c?n*". If the contents of the field fit the pattern "c?n*", then the formula returns the value True. If the field starts with anything else, the formula returns False.

Use the wildcard symbols ? and * to stand for variable characters. The ? stands for a single character. The * symbol stands for any number of characters.»

Example(s)

```
{customer.FIRST NAME} like "D?n"
```

«True, where {customer.FIRST NAME} is Dan or Don.»

```
{customer.FIRST NAME} like "D?n"
```

«False, where {customer.FIRST NAME} is Doug or Rick.»

```
{customer.LAST NAME} like "*s?n*"
```

«True, where {customer.LAST NAME} is Johnson, Olson, or Olsen.»

```
{customer.LAST NAME} like "*s?n*"
```

«False, where {customer.LAST NAME} is Johnston or Smith.»

Make Array ([,])

Usage

[x, y, z,...n]

«Build an array containing the elements x, y, z, ... n.»

Example(s)

```
[100,200,300,400]
[{file.QTYA}, {file.QTYB}, {file.QTYC}]
[({file.AMT1} *.5),({file.AMT2} *.5), ({file.AMT3} .25)]
[500,({file.QTY} /3)]
```

Note You can not have more than one data type in an array.

Make Range (to)

Usage

x to y

«Create the range x to y.»

Example(s)

```
100.00 to 250.00
```

«The range of consecutive numeric values beginning with 100.00 and ending with 250.00, including the beginning and end values.»

```
Date(1990,09,01) to Date(1990,09,20)
```

«The range of consecutive dates beginning with September 1, 1990 and ending with September 20, 1990. Both September 1 and September 20 are included in the range.»

```
"Aaron" to "Lusk"
```

«The range of consecutive string values beginning with Aaron and ending with Lusk, including the beginning and end values.»

Multiply (*)

Usage

x * y

«Multiply value x by value y.»

Example(s)

```
2883 * 1999
```

«Returns 5,763,117.»

```
{file.EXMPT} * 356.00
```

«Returns, 152,012, where Exmpt = 427.»

Negate (-())

Usage

(-(x))

«Multiply the value inside the parentheses by -1.»

Example(s)

```
-(-1)
```

«Returns 1; negative times negative = positive.»

```
-(1)
```

«Returns -1; negative times positive = negative.»

```
-({file.QTYONHND})
```

«Returns 144, where QtyOnHnd = -144.»

```
-(-(15-18))
```

«Returns -3; 15-18 = -3, -(-3)= +3, -(+3)= -3»

Not (not)

Usage

Not (x)

«Reverses the True or False value of x.»

- Not (True) = False
- Not (False) = True
- Not (Not(False)) = False
- Not (Not(True)) = True

Example(s)

```
not (A > B and B > C)
```

«If A = 5, B = 4, C = 3, the expression (A > B and B > C) is TRUE. Both conditions tied together by the Boolean operator And are TRUE, thus the entire statement has a value of TRUE. The Not operator changes the value of the expression to FALSE.»

```
not (A > B and B > C)
```

«If A = 3, B = 4, C = 3, the expression (A > B and B > C) is FALSE. One of the two conditions tied together by the Boolean operator And is FALSE, thus the entire statement has a value of FALSE. The Not operator changes the value of the expression to TRUE.»

```
not ({file.ONHAND} - {file.ORDER} > 0) = TRUE
```

«Where OnHand = 10 and Order = 11.»

```
not ({file.ONHAND} - {file.ORDER} > 0) = FALSE
```

«Where OnHand = 10 and Order = 9.»

Not Equal (<>)

Usage

x <> y

«x is not equal to y.»

The Not Equal operator tells Crystal Reports to evaluate an expression (x<>y) and return a TRUE (if x is not equal to y) or FALSE (if x is equal to y).

Example(s)

```
{order.ORDER AMOUNT} <> 400
```

«True, where Amount equals 200 or Amount equals 401, etc.»

```
{order.ORDER AMOUNT} <> 400
```

«False, where Amount equals 400.»

```
{file.DAY} <> "Thursday"
```

«True, when Day = Friday.»

```
{file.DAY} <> "Thursday"
```

«False, when Day = Thursday.»

```
{file.ONHAND} <> 0
```

«True, where the value of OnHand is 10.»

```
{file.ONHAND} <> 0
```

«False, where the value of OnHand is 0.»

```
{file.AVAILABLE} - {file.USED} <> 10
```

«True, where the value of the Available field less the value of the Used field gives a result other than 10.»

```
{file.AVAILABLE} - {file.USED} <> 10
```

«False, where the value of the Available field less the value of the Used field gives a value of 10.»

Or (or)

Usage

x or y

«Either x or y or both is true.»

- False or False = False
- False or True = True
- True or False = True
- True or True = True

Example(s)

```
If {file.CREDIT LIMIT} = 5000 or {file.SALESMAN} = "SP" Then
   {file.AMOUNT}
Else
   0
```

«This means that if the credit limit is 5000, or, if the salesman is SP (either of the conditions are true) then return the value in the Amount field, otherwise return nothing.»

```
A > B or B > C
```

«True, where A = 10, B = 6, and C = 3 (both conditions are TRUE).»

```
A > B or B > C
```

«False, where A = 5, B = 6, and C = 7 (neither of the 2 conditions are TRUE).»

```
(A > B) or (A * C - D > E) or (E / D <= B)
```

«True, where A = 5, B = 5, C = 3, D = 2, E = 12. At least one of the three conditions is true. In this case, only one condition (A * C - D > E) is true.»

Parentheses (())

Usage

(x + y) * z

«Performs the calculations inside the parentheses first.»

Parentheses are used to control the order in which the Formula Editor calculates a formula.

Example(s)

```
8 + 6 * 3 - 6 / 2 = 23
(8 + 6) * 3 - 6 / 2 = 39
(8 + 6) * (3 - 6 / 2) = 0
(8 + 6 * 3 - 6) / 2 = 10
{file.SALES} - {file.COGS} - {file.T&E} *.8
```

«Returns 11,800, where Sales = 25,000, COGS = 12,000, and T&E = 1500.»

```
{file.SALES}-(({file.COGS}-{file.T&E}) *.8)
```

«Returns 16,600, where Sales = 25,000, COGS = 12,000, and T&E = 1500.»

```
({file.SALES} - {file.COGS} - {file.T&E}) *.8
```

«Returns 9200, where Sales = 25,000, COGS = 12,000, and T&E = 1500.»

Percentage (%)

Usage

x % y

«Calculate value x as a percentage of value y [(x / y) * 100].»

Example(s)

Balance outstanding % Credit Limit means the value of Balance Outstanding is what percent of the value of Credit Limit.

```
{file.BALANCE OUTSTANDING} %
{file.CREDIT LIMIT}
```

«Returns 30.00, where Balance outstanding = $1500 and Credit Limit = $5000.»

Amount % Credit Limit means the value of Amount is what percentage of the value of Credit Limit.

```
{file.AMOUNT} % {file.CREDIT LIMIT}
```

«Returns 32.26, where Amount =2257.87 and Credit Limit =7000.»

Note If the denominator = 0, the report will be halted with a divide by zero warning. If you want to avoid this type of problem, you should put in a test similar to the following:

```
If {file.FORECAST} = 0 Then
   0
Else
   {file.SALES} % {file.FORECAST}
```

startsWith

Usage

x startsWith y

{file.FIELD} startsWith "abc"

«This operator tests to see if the contents of {file.FIELD} starts with a character string that you specify: "abc". If the contents of a field do start with the specified string, the formula returns the value True. If the field starts with anything else, the formula returns False.»

Example(s)

```
{customer.CUSTOMER NAME} startsWith "A"
```

«True, where {customer.CUSTOMER NAME} = ABC Inc.»

```
{customer.CUSTOMER NAME} startsWith "XYZ"
```

«False, where {customer.CUSTOMER NAME} = ABC Inc.»

Statement separator (;)

Usage

1 + 1; "abc";

«1 + 1 and "abc" are two different formula statements in a multiple statement formula. The semicolon between the statements specifies where one statement ends and the next one begins. Without the semicolon the statements would be treated together as an individual statement.»

Subscript ([])

Usage

x[y]

«Extract the y element from string x.»

x[y to z]

«Extract the y to z range of elements from string x.»

x[n]

«Extract the n element of array x.»

Note The subscript ranges are 1 origin; they start at 1 rather than 0.

Subscript is used to extract one or more characters from a text string or to extract an element from an array.

Example(s)

```
[100,233,466,998][3]
```

«Returns 466; 466 is the third element in the array.»

```
LASTNAME [1]
```

«Returns S, where Last Name = Smith.»

```
Postal Code [6]
```

«Returns V, where Postal Code = T5A 9V2 (the space between A and 9 counts as an element).»

```
{customer.POSTAL CODE} [5 to 7]
```

«Returns 9V2, where Postal Code = T5A 9V2.»

```
{file.ITEMNUMBER} [4 to 5]
```

«Return 40, where ItemNumber is A1/4020/B10.»

Subtract (-)

Usage

x-y

«Subtract value y from value x.»

Example(s)

```
244 - 112
```

«Returns 132.»

```
{file.SALES} - {file.COGS} - {file.S&A}
```

«Returns 214,972, where Sales = 455,031, COGS = 188,213, and S&A = 51,846.»

```
{file.ONHAND} - 877
```

«Returns 114, where OnHand = 991.»

```
Date(1991,04,15) - 12
```

«Returns Apr 03 91.»

To Currency ($)

Usage

$x

«Convert x from number to currency.»

Example(s)

The following examples all assume the following format: Decimal Places = (1.00), Negative sign = (345.00-), Currency Symbol = (Float), and Thousands Separator = (1,000.00).

```
$12345678
```

«Returns $12,345,678.00.»

```
$(123 * 456)
```

«Returns $56,088.00.»

```
$({file.QUANTITY} * 3)
```

«Returns $42.00, where Quantity = 14.»

```
$({file.MILES} * {file.PLEDGE})
```

«Returns $363.35, where Miles = 169 and Pledge = 2.15.»

Note $ * $ = error. You can not multiply a currency value by a currency value.

Variable Declarators

Usage

- NumberVar x;
- CurrencyVar x;
- StringVar x;
- BooleanVar x;
- DateVar x;

«Declares a variable x that can hold data of a type corresponding to the variable declarator used.»

Example(s)

NumberVar Amount;
«Declares a variable named Amount that can hold any value of an integer or decimal data type.»

BooleanVar Outstanding;
«Declares a variable named Outstanding that can hold a TRUE or FALSE value.»

StringVar LastName := "Adams";
«Declares a variable named LastName that can hold a string value and assigns the string "Adams" to that variable.»

CHAPTER 6

The Crystal ActiveX Control

The Crystal Report Engine

Besides acting as a powerful stand-alone report writing application, Crystal Reports provides a report generating module that you can link to and access from Visual Basic applications. As a developer using Visual Basic 5.0, you can add sophisticated report generating and printing capabilities to your applications without the time-consuming task of writing your own code.

The Crystal Report Engine is a Dynamic Link Library (DLL) that allows your applications to access the same powerful report printing features that are available in Crystal Reports. As a licensed user of Crystal Reports, you receive royalty-free rights to ship the Report Engine DLL (CRPE.DLL) with any application you create.

From your application, you access the Report Engine through the Crystal ActiveX control. When compiled, your application links with the Report Engine to add report generating functionality to your application. Reporting can be simple, producing only a single report that is sent to a printer or print window with no options available to the user, or it can be complex, allowing the user to change such things as record selection, sorting, grouping and deciding where the resulting report should be sent.

Before Using the ActiveX Control

Before you add the ActiveX control to your application, you should be familiar with some key features of the Report Engine. Review the following points, and make sure you understand each before attempting to use the ActiveX control.

- The Report Engine outputs existing reports. You can not create report files using the functionality of the Report Engine. Reports must be created using the Crystal Reports application. Make sure you understand the report creation process before trying to generate reports with the Report Engine.

- The Report Engine provides a convenient add-on, via the ActiveX control, to your existing Visual Basic application project. With just a few lines of code, you produce a powerful report writing and distribution tool that would take thousands of lines of code and weeks to produce otherwise.
- The Report Engine does not require the use of a fixed user interface. The Report Engine is designed to work with your existing development project and allows you to define the user interface your customers and users are familiar with and expect from your product.

Using the Crystal Report Engine

Any Visual Basic project that incorporates the Crystal Report Engine requires three steps:

- Create the reports your users will access.
- Design the user interface in Visual Basic that will drive the Report Engine.
- Add the Crystal ActiveX control to your application.

Creating Reports

Creating reports to include with your applications is identical to creating reports for your own use; there are no restrictions. Using the procedures outlined in the Crystal Reports manual and the Help facility, create as many kinds of reports as you want to make available to your users. You can make the reports as simple or as sophisticated as your needs dictate.

Designing the User Interface

The interface you develop to allow users to print reports is limited only by your needs and your imagination. As long as it can use the Crystal ActiveX control, it will work. The kind of user interface you select is unimportant to Crystal Reports.

Common methods of using the Report Engine include a single menu command that produces a single report, a dialog box allowing several options for printing reports, or a completely separate front-end application that is called by your application. All are acceptable techniques, and each has its advantages. How you design your user interface can depend on any or all of the following:

- the purpose of your application,
- the types of reports your application will use,
- the printing options you want to make available with those reports, and
- whether your application will offer only one report or a choice of several reports.

Consider your application and your reporting needs carefully, and design the User Interface that will use the Report Engine most efficiently.

The ActiveX Control

ActiveX is Microsoft Corporation's new technology designed to bring more powerful applications to desktops and networks. ActiveX moves beyond applications that produce static documents to a Windows environment that provides active controls, documents, and client applications that can operate and interact not only with each other, but also with network intranets and the global Internet.

ActiveX controls provide plug-in capabilities that let you add application components, and even entire applications, to your own development projects without writing more than a few lines of code. Crystal Reports supports the ActiveX concept with the Crystal ActiveX control. Use the ActiveX control to access the Crystal Report Engine and easily add all of the report processing power of Crystal Reports to your own Visual Basic applications.

Refer to your Visual Basic documentation for complete information on using ActiveX controls with Visual Basic. The rest of this chapter describes information specific to the Crystal ActiveX control.

Adding the ActiveX Control to a Project

The Crystal ActiveX control is installed in the \WINDOWS\SYSTEM directory when you install Visual Basic 5.0. You add the ActiveX control to your Visual Basic project using the Components command on the Visual Basic Project menu.

1 Open Visual Basic.

2 Open the project that you want to add the ActiveX control to.

3 Choose the Components command from the Project menu. The Custom Controls dialog box appears.

4 If Crystal Report Control 4.6 appears in the Controls tab list, click the check box next to it, click OK, and skip to step 8.

5 If Crystal Report Control does not appear in the Controls tab list, click Browse, and the Add ActiveX Control dialog box appears.

Note Crystal Report Control 4.6 is the name of the Crystal ActiveX Control when it is added to a development project. The term ActiveX control refers to a type of control, while Crystal Report Control 4.6 is the name of the Crystal ActiveX control provided by Crystal Reports. In this documentation, the terms *Crystal ActiveX control* and *Crystal Report Control* are used interchangeably to refer to the ActiveX control included with Crystal Reports for Visual Basic.

6 Use the controls in the Add Custom Control dialog box to locate and select the CRYSTL32.OCX (32-bit) file. This file is installed in your \WINDOWS\SYSTEM directory by default. Once you locate and select the file, click Open.

7 Crystal Report Control 4.6 will now appear in the Controls tab box. Click the check box next to the name of the control, and click OK.

8 Visual Basic adds the Crystal ActiveX control to your toolbox.

9 When you want to add the ActiveX control to a form, double-click the tool and the program installs it on the active form.

Using the ActiveX Control

Once you have a Crystal ActiveX control object on your form, you build the connection between your application and Crystal Reports by setting the object's properties at design time or by changing the properties at runtime. ActiveX properties let you specify:

- the name of the report you want to print in response to an application event,
- the destination for that report (Preview window, disk file, printer, or e-mail),
- the number of copies you want to print (if your report is going to the printer),
- print file information (if your report is going to a disk file),
- Preview window sizing and positioning information (if your report is going to a Preview window),
- selection formula information (if you want to limit the records in your report),
- sorting information, and
- other related properties.

Crystal ActiveX control properties can be changed either at design time or at runtime. Note, however, some properties are available only at runtime. These properties do not appear at design time.

Changing Properties at Design Time

1 Click the ActiveX control that appears on your form to select it.

2 Right-click and choose the Crystal Properties command from the shortcut menu that appears. The Property Pages dialog box appears.

3 Use the tabs and controls in this dialog box to change ActiveX control properties at design time.

Note ActiveX control properties also appear in the Visual Basic Properties box at design time.

Changing Properties at Runtime

You can set most of the ActiveX control properties at runtime by adding simple entries to your procedure code. Runtime property settings replace settings you make at design time.

Use the Action property or the PrintReport method to actually process the report at runtime. The Action property and the PrintReport method can only be used at runtime, and are the only means by which a report can actually be generated by the ActiveX control.

For more information on setting the value of specific properties at runtime, refer to the description of each property. Each description includes a code example for setting the property in your Visual Basic application.

ActiveX Properties Index

Following is a list of each ActiveX property, along with a brief description and reference to the Page on which it can be found.

Action	Action is the property that triggers the printing of the report.
BoundReportFooter	True or False property that specifies whether or not the Page Footer is to appear in the Default Bound Report.
BoundReportHeading	String property that allows the user to give a title to the report.
Connect	Logs on to a SQL server.
CopiesToPrinter	Specifies the number of copies to be printed if you are printing to a printer.
DataFiles	Specifies the location of the database files or tables used in the report.
DataSource	Specifies which data control the Crystal Reports ActiveX control is to look at for the data.
Destination	Specifies the destination to which your report is to be printed (Window, Printer, or File).
DetailCopies	Specifies the number of copies of each record in the Details section that the program is to print.
DiscardSavedData	If data is saved with the specified report, setting this property to 1 (True) discards the data.
EMailCCList	Specifies the "CC" list to which you want your E-mail message sent.

EMailMessage	Specifies the message line in your E-mail message.
EMailSubject	Specifies the subject line in your E-mail message.
EMailToList	Specifies the "To" list to which you want your E-mail message directed.
Formulas	Specifies a new string for an existing formula.
GraphData	Gets and sets the data used for a specified graph.
GraphOptions	Gets and sets a number of options for the specified graph.
GraphText	Gets and sets the various text components for the specified graph.
GraphType	Gets and sets the kind of graph used in the selected section in the specified report.
GroupCondition	Specifies what kind of change in the Group Condition Field will trigger the creation of a group.
GroupSelectionFormula	Specifies the groups to be used when printing the report.
GroupSortFields	Specifies the group field(s) that are to be used to sort your data when the report is printed.
LastErrorNumber	Returns the error code for the last runtime error.
LastErrorString	Returns the error string for the last runtime error.
Left	See Visual Basic documentation.
MarginBottom	Gets and sets the bottom margin for the specified report.
MarginLeft	Gets and sets the left margin for the specified report.
MarginRight	Gets and sets the right margin for the specified report.
MarginTop	Gets and sets the top margin for the specified report.
Name	See Visual Basic documentation.
Password	Enters the password needed to use database tables on a restricted Access .mdb file.
PrintDay	Gets and sets the day component of the print date (if different from the actual date the report is printed).

PrinterCollation	If you specify more than one copy to be printed (using the PrinterCopies property), PrinterCollation specifies whether or not the copies will be collated.
PrinterCopies	Gets and sets the number of report copies to be printed.
PrinterDriver	Gets and sets the name of the printer driver that is to print the report.
PrinterName	Gets and sets the name of the printer that is to print the report.
PrinterPort	Gets and sets the name of the printer port that is to print the report.
PrinterStartPage	Gets and sets the first page to be printed.
PrinterStopPage	Gets and sets the last page to be printed.
PrintFileCharSepQuote	Gets and sets the quote character used to enclose alphanumeric field data when printing to a file using Character Separated format.
PrintFileCharSepSeparator	Gets and sets the character(s) you want to use to separate the fields when printing to a file using the Character Separated Value format.
PrintFileName	Specifies the name of the file to which the report is to be printed.
PrintFileType	Specifies the type of the file to which the report is to be printed.
PrintFileUseRptDateFmt	When printing to a file, indicates whether or not the program should save dates in the same date format (MDY, DMY, etc.) that is used in the report or instead optimize the dates for the file format you have selected.
PrintFileUseRptNumberFmt	When printing to a file, indicates whether or not the program should print numbers in the same format (decimal places, negatives, etc.) that you have used in the report or instead optimize the numbers for the file format you have selected.
PrintMonth	Gets and sets the month component of the print date (if different from the actual date the report is printed).
PrintYear	Gets and sets the year component of the print date (if different from the actual date the report is printed).
RecordsPrinted	Determines the number of records actually printed.

RecordsRead	Determines the number of records actually read.
RecordsSelected	Determines the number of records selected.
ReportDisplayPage	Indicates which page of a multi-page report is currently being displayed in the Preview window.
ReportFileName	Specifies the report to be printed.
ReportLatestPage	Determines the last page printed in the specified report.
ReportSource	Specifies the report source as a report file or a data control.
ReportStartPage	Determines the first page printed in the specified report.
SectionFont	Specifies the font for a section.
SectionFormat	Specifies the format for a section.
SectionLineHeight	Specifies the line height for a section.
SectionMinHeight	Sets the minimum section height for the specified report section.
SelectionFormula	Specifies the records to be used when printing the report.
SessionHandle	Sets the session handle for a user once the UserName and Password properties have opened an Access .mdb file for use by the report.
SortFields	Specifies the field(s) that are to be used to sort your data when the report is printed.
SQLQuery	Gets and sets the SQL query string used by the specified report.
Status	Determines the print status for the specified report.
StoredProcParam	Gets and sets the stored procedure parameters when using a report based on SQL stored procedures.
Top	See Visual Basic documentation.
UserName	Enters the name given to a user for logging on to a protected Access .mdb file to obtain data files needed by the report.
WindowBorderStyle	Specifies the type of border for the print window.

WindowControlBox	Specifies whether or not the print window is to have a control (system menu) box in the upper left hand corner when the report is printed to a window.
WindowControls	Specifies whether or not the print controls are to appear in the Preview window when printing a report to a window.
WindowHeight	Sets the height of the print window when the report is printed to a window.
WindowLeft	Sets the distance, in pixels, that the print window is to appear from the left edge of the parent window. If the print window is a top level window, then the distance is measured from the left edge of the screen.
WindowMaxButton	Specifies whether or not the print window is to have a maximize button when the report is printed to a window.
WindowMinButton	Specifies whether or not the print window is to have a minimize button when the report is printed to a window.
WindowParentHandle	Specifies the handle of the parent window if the print window is to be the child of another window.
WindowState	Sets the state of the Preview window, normal, minimized, or maximized, when the report is printed to a Preview window.
WindowTitle	Specifies the title you want to appear in the print window title bar when the report is printed to a window.
WindowTop	Sets the distance, in pixels, that the print window is to appear from the top edge of the parent window. If the print window is a top level window, then the distance is measured from the top edge of the screen.
WindowWidth	Specifies the width of the print window in pixels.

ActiveX Methods Index

PrintReport	Triggers the printing of the report.

Crystal ActiveX Control Properties

Action

Description
Action is the property that triggers the printing of the report.

Runtime Usage
[form.]CrystalReport.Action = 1

For example:

```
CrystalReport1.Action = 1
```

«prints the specified report.»

Remarks
Set the Action property to 1 in your procedure code (CrystalReport1.Action = 1) to print the report in response to a user event.

Availability
Runtime

BoundReportFooter

Description
True or False property that specifies whether or not the Page Footer is to appear in the Default Bound Report.

Runtime Usage
[form.]CrystalReport1.BoundReportFooter = True

For example:

```
CrystalReport1.BoundReportFooter = True
```

Availability
Design time; Runtime

BoundReportHeading

Description
String property that allows the user to give the title to the report.

Runtime Usage
[form.]CrystalReport1.BoundReportHeading = "report title"

For example:

```
CrystalReport1.BoundReportHeading = "Sales by Region"
```

«prints the given title for the specified report.»

Availability
Design time; Runtime

Connect

Description
Logs on to a SQL server.

Runtime Usage
[form.]CrystalReport.Connect[= Name;UserID;Password;DatabaseQualifier$]

For example:

```
CrystalReport1.Connect = "DSN = Accounting;UID = 734;PWD =
bigboard;DSQ = Administration"
```

«connects to the "Administration" database on the "Accounting" server using the user ID #734 and the password "bigboard."»

Remarks
Enter the parameters necessary to log on to the SQL server that you need to be activated for your report.

Parameters should be in the following format:

```
DSN = name;UID = userID;PWD = password;DSQ =      database qualifier
```

- name is the server name.
- user ID is the name you have been assigned for logging onto the SQL server.
- password is the password you have been assigned for logging onto the SQL server.
- database qualifier is the database name if your server uses the database concept.

Note Before you can use this property, you must install the ODBC driver for whatever SQL database you are planning to use, and put the Database/BIN location in your path.

Note This parameter is required only when it is applicable to the ODBC driver you are using.

Availability
Design time; Runtime

CopiesToPrinter

Description
Specifies the number of copies to be printed if you are printing to a printer (if the value you assign to the Destination property is 1 - Printer).

Runtime Usage
[form.]CrystalReport.CopiesToPrinter[= NumCopies%]

For example:

```
CrystalReport1.CopiesToPrinter = 3
```

«prints three copies of the specified report.»

Remarks
The number you enter must not be a zero or a negative value.

Availability
Design time; Runtime

DataFiles

Description
Specifies the location of the database files or tables used in the report.

Runtime Usage
[form.]CrystalReport.DataFiles(ArrayIndex)[= Location$]

- Enter the file name and path of each database file or table in your report for which you want to change the location.
- Use a separate line of code for each file for which you want to change the location.
- The order of files in the array must conform to the order of files in the report. In the Crystal Reports designer, you can use the Set Location command on the Database menu to determine the order of files in the report.)
- The first file in the report is array index (0), the second file is (1), etc.

For example, to change the location of the first and third files in a report (first.dbf and third.dbf) to the c:\new directory, use the following syntax:

```
CrystalReport1.DataFiles(0) = "c:\new\first.dbf"
CrystalReport1.DataFiles(2) = "c:\new\third.dbf"
```

Remarks

- DataFiles is an array property that is available at runtime only.
- Use this property if you want to run the report with files in different locations than specified in the report.
- If you use this property, you do not have to change the locations of all files in the report. Just make certain that the array index for each file you do change matches the position of that file in the report.
- This property is cleared once the print job is printed. If you print a second time, the program reverts to the locations as originally specified in the report.

Availability
Runtime

DataSource

Description
Specifies which data control the Crystal ActiveX control is to look at for the data.

Availability
Design time only

Destination

Description

Specifies the destination to which your report is to be printed (Window, Printer, or File).

Runtime Usage

[form.]CrystalReport.Destination[= Destination%]

For example:

```
CrystalReport1.Destination = 0
```

«sends the specified report to a print window.»

Remarks

Select one of the following print destinations:

0	**0** =Window (sends the report to a print window)	crptToWindow
1	**1** =Printer (sends the report to a printer)	crptToPrinter
2	**2** =File (prints the report to a disk file for printing at a later time or for importing into other applications. If you select this property, you will also have to set the PrintFileName and PrintFileType properties.)	crptToFile
3	**3** =EMail via MAPI (sends the report to another person on your network via MAPI Email (Microsoft Mail). The report is attached to the EMail letter in the format specified by the PrintFileType property.)	crptMapi

Availability

Design time; Runtime

DetailCopies

Description

Specifies the number of copies of each record in the Details section that the program is to print.

Runtime Usage

[form.]CrystalReport.DetailCopies[= NumCopies%]

For example:

```
CrystalReport1.DetailCopies = 3
```

«Specifies that three (3) copies of each record in the details section are to be printed.»

Remarks
If DetailCopies is set to a value less than or equal to zero, the value is ignored and 1 copy of the detail section of the report is printed.

Availability
Design time, Runtime

DiscardSavedData

Description
If data is saved with the specified report, setting this property to 1 (True) discards the data.

Runtime Usage
[form.]CrystalReport.DiscardSavedData[= TrueFalse%]

For example:

```
CrystalReport1.DiscardSavedData = 1
```

«Discards the data saved with the specified report.»

Remarks
For TrueFalse% use one of the following: False = 0, True = 1.

Availability
Design time, Runtime)

EMailCCList

Description
Specifies the "CC" list to which you want your E-mail message sent.

Runtime Usage
[form.]CrystalReport.EMailCCList [= CCList$]

For example:

```
CrystalReport1.EMailCCList = "John Brown;Jane Doe"
```

«Sends a CC of the E--mail message to both John Brown and Jane Doe»

Remarks

- Applies to MAPI.
- Multiple names must be semicolon separated.

Availability
Design time, Runtime

EMailMessage

Description
Specifies the string you want to appear as the body of your E-mail message.

Runtime Usage
[form.]CrystalReport.EMailMessage[=Message$]

For example:

```
CrystalReport1.EMailMessage = "The meeting is at 4:00"
```

«Sets "The meeting is at 4:00" as the body of the E-mail message.»

Remarks
Applies to MAPI.

Availability
Design time, Runtime

EMailSubject

Description
Specifies the subject line in your E-mail message.

Runtime Usage
[form.]CrystalReport.EMailSubject[= Subject$]

For example:

```
CrystalReport1.EMailSubject = "Staff meeting"
```

«Sets "Staff meeting" as the subject line in an E-mail message.»

Remarks
Applies to MAPI.

Availability
Design time, Runtime

EMailToList

Description
Specifies the "To" list to which you want your E-mail message directed.

Runtime Usage
[form.]CrystalReport.EMailToList [=ToList$]

For example:

```
CrystalReport1.EMailToList = "Jane Doe"
```

«Makes "Jane Doe" the only name in the "To" list.»

Remarks

- Applies to MAPI.
- Multiple names must be semicolon separated.

Availability
Design time, Runtime

Formulas

Description

Specifies a new string for an existing formula.

Runtime Usage

[form.]CrystalReport.Formulas(ArrayIndex)[= "FormulaName= FormulaText"]

Enter the formula name and the string that you want to replace the existing string for each formula that you want to change in your report.

For example, to change a formula @COMMISSION to {file.SALES}*.1, and a second formula @TOTAL to {file.SALES} + {file.COMMISSION}, enter the following:

```
CrystalReport1.Formulas(0) = "COMMISSION= {file.SALES} * .1"
CrystalReport1.Formulas(1) = "TOTAL= {file.SALES} +
{file.COMMISSION}"
```

Remarks

- Formulas is an array property that is available at runtime only.
- Use a separate line of code for each formula you want to change.
- Change only those formulas that you want to change.
- The first formula you change must be assigned array index (0), the second must be assigned array index (1), etc.
- The new formula string must conform to Crystal Reports syntax requirements.
- This property is cleared once the print job is printed. If you print a second time, the program reverts to the formulas as originally specified in the report.

Note Spaces are significant in formula names. For this reason, the equal sign must follow the formula name with no intervening spaces.

Note The @ sign is not used when designating a formula name in this property.

Note You cannot use this property to create new formulas. You can only use it to change existing formulas.

Availability

Runtime

GraphData

Description

Gets and sets the data used for a specified graph.

Runtime Usage

[form.]CrystalReport.GraphData(ArrayIndex%)[= sectionCode; graphNum; row; col; field; direction$]

For example:

```
CrystalReport1.GraphData(0)= "DETAIL; 1; GROUP1; GROUP2; 0;COLANDROW"
```

«The value in Group 1 is used for the rows of the graph, the value in Group 2 is used for the columns of the graph, the first summarized field added to the report is used to set the value of the risers of the graph, and values in both columns and rows are used to create the graph.»

Remarks

With GraphData, you can specify changes to one or more graphs at runtime. Those changes then take place sequentially when you make the "Action=1" call. The array index value for GraphData simply specifies the sequence number for the change. Thus:

```
CrystalReport1.GraphData(0) = "DETAIL; 3; Group1; Group2;666;
COLANDROW"
```

when making changes to one graph only, but

```
CrystalReport1.GraphData(0) = "HEADER; 3; Group1; Group2;666;
COLANDROW"
CrystalReport1.GraphData(1) = "DETAIL; 3; Group1; Group2;666;
COLANDROW"
```

when making changes to more than one graph.

Use the following table as a guide in supplying the required values for this property:

PARAMETER	DESCRIPTION	EXPECTED VALUE
sectionCode	Specifies the section in which you want to modify a graph.	Please refer to the section code table.
GraphNum	The number of the graph within the section you want to modify.	Graphs in a section are numbered, from zero, left to right first, then top to bottom.
Row	The Group number in the report used to create rows in the graph.	GROUP1, GROUP2, GROUP3, . . ., GROUP9
col	The Group number in the report used to create columns in the graph.	GROUP1, GROUP2, GROUP3, . . ., GROUP9
field	The summarization field containing values to be used as the value of each riser in the graph.	The first summarization field added to a report is numbered 0, the second added to a report is numbered 1, etc.
direction	Whether the values in the rows, the columns, or both are used to create the graph.	ROWS, COLS, ROWANDCOL, or COLANDROW

Availability

Runtime

GraphOptions

Description

Gets and sets a number of options for the specified graph.

Runtime Usage

[form.]CrystalReport.GraphOptions(ArrayIndex%)[= sectionCode; graphNum; fontFace; barDirection;labelRisers;gridLines;legend;max;min$]

For example:

```
CrystalReport1.GraphOptions(0) ="FOOTER;0;Arial;H;T;F;X;max;min"
```

«Sets the font to Arial, sets horizontal bars, shows a data value on every riser (labelRisers = T), and toggles the grid lines off in the first Graph in the Page Footer section.»

Remarks

With GraphOptions, you can specify changes to one or more graphs at runtime. Those changes then take place sequentially when you make the "Action=1" call. The array index value for GraphOptions simply specifies the sequence number for the change. Thus:

```
CrystalReport1.GraphOptions(0) = "DETAIL; 1; Arial; H; T; F;
legend; max; min"
```

when making changes to one graph only, but

```
CrystalReport1.GraphOptions(0) = " 1; title string; subtitle string;
footnote string; series string; group string; x string;y string; z
string"
Report1.GraphOptions(1) = " 1; title string; subtitle string; footnote
string; series string; group string; x string;y string; z string"
```

when making changes to more than one graph.

Use the following chart as a guide in entering the required property values:

PARAMETER	DESCRIPTION	VALUES EXPECTED
sectionCode	Specifies the section in which you want to modify a graph.	Please refer to the section code table.
graphNum	Specifies which graph in the section you want to modify.	Graphs in a section are numbered, starting with zero, left to right first, then top to bottom.
fontFace	Specifies the font face you want to use for the entire graph.	Actual name of font, e.g., Arial
barDirection	In a bar graph, specifies the direction in which you want the graph bars to appear.	H = horizontal, V = vertical, X = as is
labelRisers	Specifies whether or not you want to show the data value on every riser.	T= true, F = False, X = as is

PARAMETER	DESCRIPTION	VALUES EXPECTED
gridLines	Specifies whether or not you want to show grid lines.	T= true, F = False, X = as is
legend	Specifies whether or not you want to show a legend.	T= true, F = False, X = as is
max	Specifies the maximum value you want included in your graph.	Enter a number.
min	Specifies the minimum value you want included in your graph.	Enter a number.

Availability
Runtime

GraphText

Description
Gets and sets the various text components for the specified graph.

Runtime Usage
[form.]CrystalReport.GraphText(ArrayIndex%)[= sectionCode; graphNum;title; subTitle; footnote; series; group; x; y; z$]

For example:

```
CrystalReport1.GraphText(0) = "HEADER; 0;;;;;;new x label;new y label;
new z label"
```

«Resets the x, y, and z labels for the first graph in the Page Header section.»

Remarks
Select your section code from the section code table.

With GraphText, you can specify changes to one or more graphs at runtime. Those changes then take place sequentially when you make the "Action=1" call. The array index value for GraphText simply specifies the sequence number for the change. Thus:

```
CrystalReport1.GraphText(0) = "DETAIL; 1; title string;subtitle
string; footnote string; series string; group string; x string;y
string;
z string"
```

when making changes to one graph only, but

```
CrystalReport1.GraphText(0) = " 1; title string; subtitle string;
footnote string; series string; group string; x string;y string; z
string"
CrystalReport1.GraphText(1) = " 1; title string; subtitle string;
footnote string; series string; group string; x string;y string; z
string"
```

when making changes to more than one graph.

"title", "subTitle", "footnote", "series", "group", "x", "y", and "z" are the strings you want to label the appropriate parts of the graph.

Availability
Runtime

GraphType

Description
Gets and sets the kind of graph used in the selected section in the specified report.

Runtime Usage
[form.]CrystalReport.GraphType(ArrayIndex%)[= sectionCode;graphNum;graphType$]

For example:

```
CrystalReport1.GraphType(0)= "GH1; 0; PIE"
```

«Specifies a Pie graph as the first graph (graphNum =0) in the Group Header 1 section.»

Remarks
With GraphType, you can specify changes to one or more graphs at runtime. Those changes then take place sequentially when you make the "Action=1" call. The array index value for GraphType specifies the sequence number for the change. Thus:

```
CrystalReport1.GraphType(0) = "DETAIL; 0; PIE"
```

when making changes to one graph only, but

```
CrystalReport1.GraphType(0) = "HEADER; 0; PIE"
CrystalReport1.GraphType(1) = "DETAIL; 0; PIE"
```

when making changes to more than one graph.

Select sectionCode from the section code table.

Graph numbers are 0 origin; the first graph in a section is number 0, the second is number 1, etc.

Multiple graphs in a section are numbered left to right first, then top to bottom.

Select from the following graph types for the graphType value for this property:

FOR THIS TYPE OF GRAPH	USE THIS CODE FOR GRAPHTYPE
Side-by-side	SIDEBYSIDE
3-D side-by-side	3DSIDE
Stacked bar	STACKEDBAR
3-D stacked bar	3DSTACKED
Percent bar	PERCENTBAR
3-D percent bar	3DPERCENT
Line	LINE
Area	AREA
3-D bars	3DBARS
Pie	PIE
Multiple pie	MULTIPLEPIE
Weighted pie	WEIGHTEDPIE

Availability
Runtime

GroupCondition

Description
Specifies what kind of change in the Group Condition Field will trigger the creation of a group.

Runtime Usage
[form.]CrystalReport.GroupCondition(SequentialIndex%)[= group; field; condition; sortDirection$]

For example:

```
CrystalReport1.GroupCondition(0)="GROUP1;{header.ordernum};ANYCHANGE;A
"
```

«Specifies that any change in the ordernum field in Group1 will trigger a new grouping.»

Remarks

Refer to the following tables for parameter values for this property:

PARAMETER	DESCRIPTION	VALUES EXPECTED
group	The group in which you want to set the group condition.	The outermost group on the report is GROUP1, the next group is GROUP2, etc.
field	The name of the field that triggers a grouping whenever its value changes.	Enter the name in the following format: {table.fieldname}
condition	Enter the condition that triggers the grouping.	See the tables below.
sortDirection	The direction in which groups are to be sorted.	A = Ascending, D = Descending

Date Field Conditions

CONDITION	CONDITION CODE
Daily	DAILY
Weekly	WEEKLY
Bi-weekly	BIWEEKLY
Semi-monthly	SEMIMONTHLY
Monthly	MONTHLY
Quarterly	QUARTERLY
Semi-annually	SEMIANNUALLY
Annually	ANNUALLY

Boolean Field Conditions

CONDITION	CONDITION CODE
To Yes	TOYES
To No	TONO
Every Yes	EVERYYES
Every No	EVERYNO
Next Is Yes	NEXTISYES
Next Is No	NEXTISNO

Condition for all other data types

CONDITION	CONDITION CODE
Any Change	ANYCHANGE

Availability

Runtime

GroupSelectionFormula

Description

Specifies the groups to be used when printing the report.

Runtime Usage

[form.]CrystalReport.GroupSelectionFormula[= "GroupSelectionFormula"]

Enter the group selection formula just as you would enter it in the Formula Editor. For example, to limit your report to those groups with a subtotal on the {header.AMOUNT field} less than $10,000 (with subtotals triggered by changes in the {header.CUSTNUMB} field), you would enter the following as a group selection formula:

```
CrystalReport1.GroupSelectionFormula = Sum ( {header.AMOUNT},
{header.CustNumb}) < 10000
```

Remarks

If your group selection formula includes internal quotes, change all of the internal double quotes to single quotes and then surround the entire selection formula in double quotes.

Note If you have created a group selection formula in your report at design time, any group selection formula you enter here will be appended to that group selection formula, connected by an "and". Thus, your records will be selected based on a combination of the two formulas.

Availability

Design time; Runtime

GroupSortFields

Description

Specifies the group field(s) that are to be used to sort your data when the report is printed.

Runtime Usage

[form.]CrystalReport.GroupSortFields(ArrayIndex)[= "{+|-}GroupField"]

Enter the group field(s) on which you want your report to be sorted.

For example, assume that you have broken your data into state groups and had Crystal Reports count the number of customers in each group. In order to print the group with the highest count first, then the group with the next highest count, etc. (descending order), you would enter a string similar to the following:

```
CrystalReport1.GroupSortFields(0) = "-Count({customer.CUSTOMER},
{customer.STATE})"
```

Remarks

- GroupSortFields is an array property available at runtime only.
- Use a separate line of code to specify each group sort field.
- Enter group sort fields in the order that you want them to sort your report. For example, if you want your report to be sorted first on group sort field A and then on group sort field B, specify group sort field A in your first line of code and group sort field B in your second line of code.
- The first group sort field you specify must be assigned array index 0, the second group sort field must be assigned array index 1, etc.
- The index values you assign must be continuous; no gaps are allowed (0,1,2 = OK, 0,1,3 = wrong)
- Array index values must be subscripted in the code immediately after the property name, i.e., CrystalReport1.GroupSortFields(0) =.
- If you have specified sort fields for your report at design time, any sort fields you enter here will replace the sort fields in your report.
- If you don't use this property, the program will use the sorting instructions that you specified in the report.
- If you want to clear the group sort fields in your report, use an empty string (CrystalReport1.GroupSortFields(0) = "")
- This property is cleared once the print job is printed. If you print a second time, the program reverts to the group sort fields as originally specified in the report.

Note The group sort field entry must follow the sort direction sign (+ or -) with no intervening space.

Note To find the correct syntax for any group in your report using Crystal Reports for Visual Basic: select Insert|Formula field, enter any formula name in the Insert Formula Dialog box when it appears, click the scroll button on the Fields list in the Formula Editor when it appears, and double-click the group field of interest.

Crystal Reports enters the group field name in the Formula Text box. Use the name and syntax from that text box when constructing your group sort field string.

Availability
Runtime

LastErrorNumber

Description
Returns the error code for the last runtime error.

Runtime Usage
[form.]CrystalReport.LastErrorNumber

For example:

```
'If error occurs, go to Error Handler ErrorHandler:MsgBox
CrystalReport1.LastErrorNumber
```

«if an error occurs, this code calls up a message box that displays the error number.»

Remarks
LastErrorNumber is a runtime--only property.

Note LastErrorNumber must come after the Action call in order to display relevant values. After you have printed your report, you can refer to this property to get an error number (if any). If there was no error in printing, LastErrorNumber = 0.

Availability
Runtime (read and write)

LastErrorString

Description
Returns the error string for the last runtime error.

Runtime Usage
[form.]CrystalReport.LastErrorString

For example:

```
'If error occurs, go to Error HandlerErrorHandler:MsgBox
CrystalReport1.LastErrorString
```

«if an error occurs, this code calls up a message box that displays the error string.»

Remarks
LastErrorString is a runtime-only property.

Note LastErrorString must come after the Action call in order to display relevant values. After you have printed your report, you can refer to this property to get an error string (if any). If there was no error in printing, LastErrorNumber = 0.

Availability
Runtime

MarginBottom

Description
Sets the bottom margin for the specified report.

Runtime Usage
[form.]CrystalReport.MarginBottom[=MarginSetting%]

For example:

```
CrystalReport1.MarginBottom = 720
```

«Sets a 1/2 inch bottom margin for the report (1 inch = 1440 twips)»

Remarks
MarginSetting% is the margin you want, in twips.

Availability
Design time. Runtime

MarginLeft

Description
Sets the left margin for the specified report.

Runtime Usage
[form.]CrystalReport.MarginLeft[=MarginSetting%]

For example:

```
CrystalReport1.MarginLeft = 1440
```

«Sets a 1 inch left margin for the report (1 inch = 1440 twips)»

Remarks
MarginSetting% is the margin you want, in twips.

Availability
Design time, Runtime)

MarginRight

Description
Sets the right margin for the specified report.

Runtime Usage
[form.]CrystalReport.MarginRight[=MarginSetting%]

For example:

```
CrystalReport1.MarginRight=1440
```

«Sets a 1 inch right margin for the report (1 inch = 1440 twips)»

Remarks
MarginSetting% is the margin you want, in twips.

Availability
Design time, Runtime

MarginTop

Description
Sets the top margin for the specified report.

Runtime Usage
[form.]CrystalReport.MarginTop[=MarginSetting%]

For example:

```
CrystalReport1.MarginTop = 720
```

«Sets a 1/2 inch top margin for the report (1 inch = 1440 twips)»

Remarks
MarginSetting% is the margin you want, in twips.

Availability
Design time, Runtime

Password

Description
Enters the password needed to use database tables on a restricted Access .mdb file.

Runtime Usage
[form.]CrystalReport.Password[= Password$]

For example:

```
CrystalReport1.Password = "dogsncats"
```

«enters the password "dogsncats"»

Remarks
Enter the password you have been assigned.

Availability
Design time; Runtime

PrintDay

Description
Sets the day component of the print date (if different from the actual date the report is printed).

Runtime Usage
[form.]CrystalReport.PrintDay[=Day%]

For example:

```
CrystalReport1.PrintDay = 23
```

«Sets 23 as the print day.»

Remarks

- Enter a value from 1 to 31
- The PrintYear, PrintMonth, and PrintDay properties function together. You must change the value of all three to change the print date. If you do not change all three, the print date saved with the report is used. This may be the current date if a specific date is not saved with the report.

Availability
Design time and Runtime

PrinterCollation

Description
If you specify more than one copy to be printed (using the PrinterCopies property), PrinterCollation specifies whether or not the copies will be collated.

Runtime Usage
[form.]CrystalReport.PrinterCollation[=CollationCode%]

For example:

```
CrystalReport1.PrinterCollation = 1
```

«Collates the copies of the specified report»

Remarks
Select your CollationCode% value from the following table:

STATUS	CODE	CONSTANTS
Uncollated	0	crptUncollated
Collated	1	crptCollated
Default Collation	2	crptDefault

Availability
Design time, Runtime

PrinterCopies

Description
Sets the number of report copies to be printed.

Runtime Usage
[form.]CrystalReport.PrinterCopies[=NumCopies%]

For example:

```
CrystalReport1.PrinterCopies = 3
```

«Specifies that the program is to print three (3) copies of the report.»

Remarks
The number you use for PrinterCopies must not be zero or a negative value.

Availability
Design time, Runtime

PrinterDriver

Description
Gets and sets the name of the printer driver that is to print the report.

Runtime Usage
[form.]CrystalReport.PrinterDriver[= DriverName$]

For example:

```
CrystalReport1.PrinterDriver = "Epson24.drv"
```

«Sets the printer driver to be the Epson 24 pin driver»

Remarks
The PrinterDriver, PrinterName, and PrinterPort properties work together to define the printer that the report is to be sent to. All three properties must be set in order to define a new printer. If all three properties are not set, the printer defined in the report will be used. This may be the users default printer if none has been specified in the report.

Availability
DesignTime, Runtime

PrinterName

Description
Gets and sets the name of the printer that is to print the report.

Runtime Usage
[form.]CrystalReport.PrinterName[= PrinterName$]

For example:

```
CrystalReport1.PrinterName= "Epson LQ-850"
```

«Specifies the Epson LQ-850 printer»

Remarks
The PrinterDriver, PrinterName, and PrinterPort properties work together to define the printer that the report is to be sent to. All three properties must be set in order to define a new printer. If all three properties are not set, the printer defined in the report will be used. This may be the users default printer if none has been specified in the report.

Availability
Design time, Runtime

PrinterPort

Description
Sets the port for the specified printer.

Runtime Usage
[form.]CrystalReport.PrinterPort[= PortName$]

For example:

```
CrystalReport1.PrinterPort= "LPT1"
```

«Sets the printer port to LPT1»

Remarks
The PrinterDriver, PrinterName, and PrinterPort properties work together to define the printer that the report is to be sent to. All three properties must be set in order to define a new printer. If all three properties are not set, the printer defined in the report will be used. This may be the users default printer if none has been specified in the report.

Availability
Design time, Runtime

PrinterStartPage

Description
Sets the first page to be printed.

Runtime Usage
[form.]CrystalReport.PrinterStartPage[= StartPage%]

For example:

```
CrystalReport1.PrinterStartPage = 7
```

«Specifies that printing is to begin with page 7 of the report.»

Remarks
If a value less than or equal to 0 is used for PrinterStartPage, the value is ignored and printing starts with page 1.

Availability
Design time, Runtime

PrinterStopPage

Description
Sets the last page to be printed.

Runtime Usage
[form.]CrystalReport.PrinterStopPage[=StopPage%]

For example:

```
CrystalReport1.PrinterStopPage = 12
```

«Specifies that the printing is to end with **page** 12 of the report.»

Remarks
Use a value of -1 for PrinterStopPage to indicate that printing is to continue through to the last page.

Availability
Design time, Runtime

PrintFileCharSepQuote

Description
Gets and sets the quote character used to enclose alphanumeric field data when printing to a file using Character Separated format.

Runtime Usage
[form.]CrystalReport.PrintFileCharSepQuote[=Quote$]

For example:

```
CrystalReport1.PrintFileCharSepQuote = " ' "
```

«Uses the quotation character ' to surround values saved in character separated format.»

Remarks

- Applies only when PrintFileType is 5 - Character-separated values.
- Applies only when Destination is 2 - File, or 3 - EMail to MAPI.
- If you assign a string to PrintFileCharSepQuote that is longer than one character, the control uses only the first character of that string. For example, if you assign "quote" to the property, the VBX will only recognize "q".

Availability
Design time, Runtime

PrintFileCharSepSeparator

Description
Sets the character(s) you want to use to separate the fields when printing to a file using the Character Separated Value format.

Runtime Usage
[form.]CrystalReport.PrintFileCharSepSeparator[=Separator$]

For example:

```
CrystalReport1.PrintFileCharSepSeparator= "@"
```

«Specifies that the "@" character is to be used for separating field values.»

Remarks

- Applies only when PrintFileType is 5 - Character-separated values.
- Applies only when Destination is 2 - File, or 3 - EMail to MAPI.

Availability
Design time and Runtime

PrintFileName

Description

Specifies the name of the file to which the report is to be printed.

Runtime Usage

[form.]CrystalReport.PrintFileName[= FileName$]

For example:

```
CrystalReport1.PrintFileName ="c:\crw\cust_rpt.txt"
```

«prints the report to a file named "cust_rpt.txt" in the c:\crw directory.»

Remarks

- You can double-click this property or click the Ellipsis in the Settings box to call up the Choose Print Filename dialog box. In that dialog box, select the name of the file and the path to which you want the program to print the report.
- Select a value for this property only if you are printing to a file (if the value you assigned to the Destination property is 2 - File).

Note If you want to specify the PrintFileName at runtime, make certain that you enclose it in quotes in your code.

Availability

Design time; Runtime

PrintFileType

Description

Specifies the type of print file used when printing a report to a file.

Runtime Usage

[form.]CrystalReport.PrintFileType[= FileType%]

For example:

```
CrystalReport1.PrintFileType = 1
```

«prints the report to a file in a tab separated format.»

Remarks

Select one of the following print file types if you are printing to a file (if the value you assigned to the Destination property is 2 - File).

0 - Record	Record style (columns of values). Doesn't use commas or separators. Outputs every record with a fixed field width.	crptRecord
1 - Tab separated	Tab separated values. Presents data in tabular form. Encloses alphanumeric field data in quotes and separates fields with tabs.	crptTabSep
2 - Text	Text style. Saves the data in ASCII text format with all values separated by spaces. This style looks most like the printed page.	crptText
3 - DIF	Saves the data in DIF (data interchange format) format. This format is often used for the transfer of data between different spreadsheet programs.	crptDIF
4 - CSV	Comma separated values. Encloses alphanumeric field data in quotes and separates fields with commas.	crptCSV
5 - Character Separated	Saves the data as character separated values in ASCII text format. All values are separated by a character or characters specified by the PrintFileCharSepSeparator property.	crptCharSep
6 - Tab separated Text	Saves the data in ASCII text format with all values separated by tabs.	crptTabSepText
7 - Crystal Reports RPT	Standard Crystal Reports RPT format is used. Most often used for sending the report to another user via EMail.	crptCrystal
8 - Excel 2.1 XLS	Exports the report as a Microsoft Excel 2.1 Worksheet.	crptExcel21

9 - Excel 3.0 XLS	Exports the report as a Microsoft Excel 3.0 Worksheet.	crptExcel30
10 - Excel 4.0 XLS	Exports the report as a Microsoft Excel 4.0 Worksheet.	crptExcel40
11 - Lotus 1-2-3 WK1	Exports the report as a Lotus 1-2-3 WK1 format worksheet.	crpt123wk1
12 - Lotus 1-2-3 WK3	Exports the report as a Lotus 1-2-3 WK3 format worksheet.	crpt123wk3
13 - Lotus 1-2-3 WKS	Exports the report as a Lotus 1-2-3 WKS format worksheet.	crpt123wks
15 - RTF	Saves the data in Rich Text Format.	crptRTF
17 - Word for Windows	Uses the Microsoft Word for Windows format to save the data in the report.	crptWinWord
19 - Excel 5.0 XLS	Exports the report as a Microsoft Excel 5.0 Worksheet.	crptExcel50
20 - HTML 3.0	Exports the report in HTML format.	crptHTML30
21 - Internet Expl	Exports the report in Internet Explorer format.	crptIntExpl
22 - Netscape	Exports the report in Netscape format.	crptNetscape

Availability
Design time; Runtime

PrintFileUseRptDateFmt

Description
When printing to a file, indicates whether or not the program should save dates in the same date format (MDY, DMY, etc.) that is used in the report or instead optimize the dates for the file format you have selected.

Runtime Usage
[form.]CrystalReport.PrintFileUseRptDateFmt[= TrueFalse%]

For example:

```
CrystalReport1.PrintFileUseRptDateFmt = 1
```

«Specifies that the program should print dates in the same format as used in the report.»

Remarks

- Applies only when PrintFileType is 0 - Record, 1 - Tab-separated, 3 - DataInterchangeFormat (DIF), 4 - CSV, or 5 - Character Separated.
- Applies only when Destination is 2 - File or 3 - EMail to MAPI.
- For TrueFalse%, use one of the following values: False = 0, True = 1.

Availability
Design time, Runtime

PrintFileUseRptNumberFmt

Description
When printing to a file, indicates whether or not the program should print numbers in the same format (decimal places, negatives, etc.) that you have used in the report or instead optimize the numbers for the file format you have selected.

Runtime Usage
[form.]CrystalReport.PrintFileUseRptNumberFmt[=TrueFalse%]

For example:

```
CrystalReport1.PrintFileUseRptNumberFmt = 1
```

«Specifies that the program should print numbers in the same format as used in the report.»

Remarks

- Applies only when PrintFileType is 0 - Record, 1 - Tab-separated, 3 - DataInterchangeFormat (DIF), 4 - CSV, or 5 - Character Separated
- Applies only when Destination is 2 - File or 3 - EMail to MAPI.
- For TrueFalse%, use one of the following values: False = 0, True = 1.

Availability
Design time, Runtime

PrintMonth

Description
Sets the month component of the print date (if different from the actual date the report is printed).

Runtime Usage
[form.]CrystalReport.PrintMonth[= Month%

For example:

```
CrystalReport1.PrintMonth= 7
```

«Sets July as the print month.»

Remarks

- Enter a value from 1-12 with January = 1, December = 12.
- The PrintYear, PrintMonth, and PrintDay properties function together. You must change the value of all three to change the print date. If you do not change all three, the print date saved with the report is used. This may be the current date if a specific date is not saved with the report.

Availability
Design time, Runtime

PrintYear

Description
Sets the year component of the print date (if different from the actual date the report is printed).

Runtime Usage
[form.]CrystalReport.PrintYear[=Year%]

For example:

```
CrystalReport1.PrintYear = 1994
```

«Sets the year component of the print date to 1994.»

Remarks

- Enter the print year as a four-digit number.
- The PrintYear, PrintMonth, and PrintDay properties function together. You must change the value of all three to change the print date. If you do not change all three, the print date saved with the report is used. This may be the current date if a specific date is not saved with the report.

Availability
Design time, Runtime

RecordsPrinted

Description
Determines the number of records actually printed.

Runtime Usage
[form.]CrystalReport.RecordsPrinted

For example:

```
Printed& = CrystalReport1.RecordsPrinted
```

«Fetches the number of records printed and stores it in the Printed variable.»

Remarks
If the report being printed contains one or more group selection formulas, the value returned by RecordsPrinted maybe significantly less than the value returned by RecordsSelected. Otherwise, this value should equal RecordsSelected.

Availability
Runtime

RecordsRead

Description
Determines the number of records actually processed.

Runtime Usage
[form.]CrystalReport.RecordsRead

For example:

```
Read% = CrystalReport1.RecordsRead
```

«Fetches the number of records read and saves it in the Read variable.»

Remarks
If the Crystal Reports Engine generates a SQL query to obtain data from a SQL database when the report is printed, RecordsRead will only return the number of records received by the Print Engine from the query. This value may be significantly smaller than the number of records actually in the SQL database table.

Availability
Runtime

RecordsSelected

Description
Determines the number of records selected for inclusion in the report out of the total number of records read.

Runtime Usage
[form.]CrystalReport.RecordsSelected

For example:

```
Selected& = CrystalReport1.RecordsSelected
```

«Fetches the number of records selected and saves it in the Selected variable.»

Remarks
RecordsSelected will return a value anywhere between zero and the value returned by RecordsRead. The value returned by RecordsSelected depends on the queries and selection formulas set up in the report.

Availability
Runtime

ReportDisplayPage

Description
Indicates which page of a multi-page report is currently being displayed in the Preview window.

Runtime Usage
[form.]CrystalReport.ReportDisplayPage

For example:

```
Result% = CrystalReport1.DisplayPage
```

«Fetches the number of the displayed **page** and stores it in the Result variable.»

Availability
Runtime

ReportFileName

Description
Specifies the report to be printed.

Runtime Usage
[form.]CrystalReport.ReportFileName[= ReportName$]

For example:

```
CrystalReport1.ReportFileName = "c:\crw\company.rpt"
```

«prints the report named "company.rpt" that is located in the c:\crw directory.»

Remarks
You can double-click this property or click the Ellipsis in the Settings box to call up the Choose Report File dialog box. In that dialog box, select the name and path of the report you want the program to print in response to a CRW Custom Control event.

Note If you want to specify the ReportFileName at runtime, make certain that you enclose it in quotes in your code.

Availability
Design time; Runtime

ReportLatestPage

Description
Determines the last page printed in the specified report.

Runtime Usage
[form.]CrystalReport.ReportLatestPage

For example:

```
Latest% = CrystalReport1.ReportLatestPage
```

«Fetches the number of the last page printed and stores it in the Latest variable.»

Availability
Runtime

ReportSource

Description
Specifies where the report template will come from.

Runtime Usage
[form.]CrystalReport1.ReportSource = [TrueFalse%]

For example:

```
CrystalReport1.ReportSource = 1
```

Availability
Design time; Runtime

ReportStartPage

Description
Determines the first page printed in the specified report.

Runtime Usage
[form.]CrystalReport.ReportStartPage

For example:

```
StartPage% = CrystalReport1.ReportStartPage
```

«Fetches the number of the first page printed and stores it in the StartPage variable.»

Availability
Runtime

SectionFont

Description
Sets the font for one or more sections in the specified report.

Runtime Usage
[form.]CrystalReport.SectionFont(SequentialIndex%)[= sectionCode; fontName; size; italic; bold;underline;strikethru$]

For example:

```
CrystalReport1.SectionFont(0)="Footer;Arial;12;N;N;N;Y"
```

«Sets the font for the footer section to 12 point, Arial, strikethru.»

Remarks

With SectionFont, you can specify changes to one or more sections at runtime. Those changes then take place sequentially when you make the "Action=1" call. The array index value for SectionFont simply specifies the sequence number for the change. Thus:

```
CrystalReport1.SectionFont(0)= "DETAIL;Arial;12;N;N;N;Y"
```

when making changes to the DETAIL section only, but

```
CrystalReport1.SectionMinHeight(0) = "HEADER;Arial;12;N;N;N;Y"
CrystalReport1.SectionMinHeight(1) = "DETAIL;Arial;12;N;N;N;Y"
```

when making changes to more than one section.

Use the following table as a guide in supplying the required values for this property:

PARAMETER	DATA TYPE	VALUE EXPECTED
sectionCode	string	Please refer to the section code table.
FontName	string	The actual font name, e.g. Arial or Helvetica
size	number	The size of the font in points, e.g., 12 or 16
italic	character	T = true, F = False, X = as is1
bold	character	T = true, F = False, X = as is1
underline	character	T = true, F = False, X = as is1
strikethru	character	T = true, F = False, X = as is1 1X (as is) uses the value saved with the report

Availability

Runtime

SectionFormat

Description

Sets the format for one or more sections in the specified report.

Runtime Usage

[form.]CrystalReport.SectionFormat(SectionArrayIndex%)[= sectionCode; visible; newPageBefore; newPageAfter; keepTogether; suppressBlank; resetPageNAfter; printAtPageBottom$]

For example:

```
CrystalReport1.SectionFormat(0)= "GH2;F;X;X;X;X;X;X"
```

«Hides the Group Header 2 section (visible = F) while maintaining default settings for all other switches»

Remarks

With SectionFormat, you can specify changes to one or more sections at runtime. Those changes then take place sequentially when you make the "Action=1" call. The sequential index value for SectionFormat simply specifies the sequence number for the change. Thus:

```
CrystalReport1.SectionFormat(0)= "DETAIL;T;F;F;X;X;X;X"
```

when making changes to the DETAIL section only, but

```
CrystalReport1.SectionFormat(0) = "HEADER;T;F;F;X;X;X;X"
CrystalReport1.SectionFormat(1) = "DETAIL;T;F;F;X;X;X;X"
```

when making changes to more than one section.

Use the following table as a reference when entering parameter values for this property:

PARAMETER	EXPECTED VALUE
sectionCode	Please refer to the section code table.
Visible	T = true, F = False, X = as is1
newPageBefore	T = true, F = False, X = as is1
newPageAfter	T = true, F = False, X = as is1
keepTogether	T = true, F = False, X = as is1
suppressBlank	T = true, F = False, X = as is1
resetPageNAfter	T = true, F = False, X = as is1
printAtPageBottom	T = true, F = False, X = as is1 1X (as is) uses the settings saved with the report

Availability

Runtime

SectionLineHeight

Description

Specifies the line height in twips. A twip is 1/1440 inch; there are 20 twips in a point.

Runtime Usage

[form.]CrystalReport.SectionLineHeight(SequentialIndex%)[= sectionCode; line; height; ascent$]

For example:

```
CrystalReport1.SectionLineHeight(0) = "GH0; 1; 500; 300"
```

«Sets the line height for the second line in the group header zero section to a height of 500 twips with an ascent of 300 twips»

Remarks

With SectionLineHeight, you can specify changes to one or more sections at runtime. Those changes then take place sequentially when you make the "Action=1" call. The sequential index value for SectionLineHeight simply specifies the sequence number for the change. Thus:

```
CrystalReport1.SectionLineHeight(0)= "DETAIL;1;500;300"
```

when making changes to the DETAIL section only, but

```
CrystalReport1.SectionLineHeight(0) = "HEADER;1;500;300"
CrystalReport1.SectionLineHeight(1) = "DETAIL;1;500;300"
```

when making changes to more than one section.

Use the following table as a guide in supplying the required values for this property:

sectionCode	Specifies the section code for the report section(s) from which you want to set a new line height.
LineN	Specifies the line(s) for which you want to set the line height. Line numbers in a section are 0 origin: the first line number is 0, the second is 1, etc.
height	Specifies the line height in twips. A twip is 1/1440 inch; there are 20 twips in a point.
Ascent	Specifies the ascent in twips. Ascent is the distance from the top of the allotted line space (line height) to the baseline of the font. The ascent parameter is used to specify the position of the baseline if you specify an oversized or undersized line height. If you set ascent to 0, the program puts the baseline at the top of the space; if you set ascent to the same value as height, the program sets the baseline at the bottom of the space. For any other baseline, specify the ascent in twips.

Availability

Runtime

SectionMinHeight

Description

Sets the minimum section height for the specified report section.

Runtime Usage

[form.]CrystalReport.SectionMinHeight(SequentialIndex%)[=sectionCode;minHeight$]

For example:

```
CrystalReport1.SectionMinHeight(0) = "ALL; 500"
```

«Sets the minimum height for all sections to 500 twips.»

Remarks
With SectionMinHeight, you can specify changes to one or more sections at runtime. Those changes then take place sequentially when you make the "Action=1" call. The array index value for SectionMinHeight simply specifies the sequence number for the change. Thus:

```
CrystalReport1.SectionMinHeight(0)= "DETAIL;500"
```

when making changes to the DETAIL section only, but

```
CrystalReport1.SectionMinHeight(0) = "HEADER;500"
CrystalReport1.SectionMinHeight(1) = "DETAIL;500"
```

when making changes to more than one section.

Availability
Runtime

SelectionFormula

Description
Specifies the records to be used when printing the report.

Runtime Usage
[form.]CrystalReport.SelectionFormulaProperty[= SelectionFormula$]

Enter the selection formula just as you would enter it in the Formula Editor in Crystal Reports. For example, to include only those records that have a quantity bigger than 5 in the {file.Qty} field, you would enter "{file.QTY} > 5" as your selection formula, i.e., CrystalReport1.SelectionFormula = "{file.QTY} > 5"

Remarks

- Make certain that you enclose your selection formula in double quotes.
- If your selection formula includes internal quotes, for example:

  ```
  {file.STATE} = "CA"
  ```

  change all of the internal double quotes to single quotes and then surround the entire selection formula in double quotes like this:

  ```
  "{file.STATE} = 'CA'"
  ```

- If you have created a selection formula in your report at design time, any selection formula you enter here will be appended to that selection formula. Thus, your records will be selected based on a combination of the two selection formulas.

Availability
Design time; Runtime

SessionHandle

Description
Sets the session handle for a user once the UserName and Password properties have opened an Access .mdb file for use by the report.

Runtime Usage
[form.]CrystalReport.SessionHandle[= Handle%]

Remarks
If you have already opened a Jet session in your Visual Basic application, you can set this property to be the current session handle. Otherwise you will have to use the Password and UserName properties to establish the Jet session.

For example:

```
CrystalReport1.SessionHandle = CurrentSessionHandle
```

«sets the session handle to the session handle returned elsewhere in the application and stored in the variable CurrentSessionHandle»

Availability
Runtime

SortFields

Description
Specifies the field(s) that are to be used to sort your data when the report is printed.

Runtime Usage
[form.]CrystalReport.SortFields(ArrayIndex)[= "{+|-}SortField"]

Enter the fields on which you want the data in your report to be sorted.

For example, to sort an order database alphabetically by customer, and then by order date, you can enter code similar to this:

```
CrystalReport1.SortFields(0) =
"+{orders.CUSTOMER}"CrystalReport1.SortFields(1) =
"+{orders.ORDERDATE}"
```

Remarks

- SortFields is an array property available only at runtime.
- Use a separate line of code to specify each sort field.
- Enter sort fields in the order that you want them to sort your report. For example, if you want your report to be sorted first on field A and then on field B, specify sort field A in your first line of code and sort field B in your second line of code.
- The sort field you specify must be assigned array index 0, the second sort field must be assigned array index 1, etc.
- The index values you assign must be continuous; no gaps are allowed (0,1,2 = OK, 0,1,3 = wrong).

- Array index values must be subscripted in the code immediately after the property name, i.e., CrystalReport1.SortFields(0) =.
- If you have specified sort fields for your report at design time, any sort fields you enter here will replace the sort fields in your report.
- If you don't use this property, the program will use the sorting instructions that you specified in the report.
- If you want to clear the sort fields in your report, use an empty string (CrystalReport1.SortFields(0) = "").
- Enclose field names in braces.
- Sort fields can be database fields or formula fields. If you sort on a formula field, use the @ sign before the formula name, i.e., @FORMULANAME.

Availability
Runtime

SQLQuery

Description
Sets the SQL query string used by the specified report.

Runtime Usage
[form.]CrystalReport.SQLQuery[=SQLQuery$]

For example:

```
CrystalReport1.SQLQuery = "SELECT authors.au_id, authors.au_lname,
authors.au_fname FROM pubs2.dbo.authors authors WHERE authors.au_
lname >  'Madison'"
```

«Queries the SQL database to return only the records where the authors last name falls after Madison alphabetically.»

Remarks
You may only change the WHERE and FROM sections of an SQL query. Although the property requires that you enter the entire SQL query, the SELECT section must not be different from the original query.

Availability
Design time, Runtime

Status

Description
Determines the print status for the specified report.

Runtime Usage
[form.]CrystalReport.Status

For example:

```
Status% = CrystalReport1.Status
```

«Fetches the print status and saves it to the Status variable.»

Remarks
The Status property will return one of the following values:

- 0 - The report has not been printed or has not finished printing.
- 3 - The report has finished printing.
- 5 - The report has been canceled by the user.

Availability
Runtime

StoredProcParam

Runtime Usage
[form.]CrystalReport.StoredProcParam(Parameter Array Index%)[= newParameter$]

For example:

```
CrystalReport1.StoredProcParam(0) = "06/14/1989"
```

«Sets the first stored procedure parameter to the date June 14, 1989»

Remarks
StoredProcParam sets the value of the specified parameter in a SQL database table that is based on a stored procedure. Pass the value you wish to set the parameter to as a string. If the parameter expects a different data type, you still must pass the value as a string. For example, to pass the integer value 396, use the string "396". The Crystal Reports Engine will handle converting the value into integer format.

Availability
Runtime only. Sets the stored procedure parameters when using a report based on SQL stored procedures.

UserName

Description
Enters the name given to a user for logging on to a protected Access .mdb file to obtain data files needed by the report.

Runtime Usage
[form.]CrystalReport.UserName[= Name$]

For example:

```
CrystalReport1.UserName = "MIS"
```

«enters the user name "MIS"»

Remarks

- Enter the name you have been assigned.
- The name must be enclosed in quotes.

Availability
Runtime

WindowBorderStyle

Description

Specifies the type of border for the print window.

Runtime Usage

[form.]CrystalReport.WindowBorderStyle[= BorderStyle%]

For example:

```
CrystalReport1.WindowBorderStyle = 2
```

«sets a sizable border style (Style #2) for the print window»

Remarks

Select one of the following border styles for the print window:

0	**0** -None (creates a window with no border).	**crptNoBorder**
1	**1** -Fixed Single (creates a window of a fixed size with a single line border).	**crptFixedSingle**
2	**2** -Sizable (creates a window that can be resized by the user).	**crptSizeable**
3	**3** -Fixed Double (creates a window of fixed size with a double line border).	**crptFixedDouble**

Availability

Design time; Runtime

WindowControlBox

Description

Specifies whether or not the print window is to have a control (system menu) box in the upper left hand corner when the report is printed to a window.

Runtime Usage

[form.]CrystalReport.WindowControlBox[= {True|False}]

For example:

```
CrystalReport1.WindowControlBox = True
```

«specifies that a control box (system menu) is to appear in the print window.»

Remarks

- Select True if you want the window to contain a control box. Select False if you don't.
- Select a value here only if you are printing to a window (if Destination = 0).

Availability

Design time; Runtime

WindowControls

Description
Specifies whether or not the print controls are to appear in the Preview window when printing a report to a window.

Runtime Usage
[form.]CrystalReport.WindowControls[=TrueFalse%]

For example:

```
CrystalReport1.WindowControls = 1
```

«Specifies that print controls are to appear in the Preview window.»

Remarks
For TrueFalse% use one of the following: False = 0, True = 1.

Availability
Design time, Runtime

WindowHeight

Description
Sets the height of the print window when the report is printed to a window.

Runtime Usage
[form.]CrystalReport.WindowHeight[= Height%]

For example:

```
CrystalReport1.WindowHeight = 300
```

«sets the height of the print window to 300 pixels.»

Remarks

- If you are not satisfied with the default settings, enter the external height you want for your print window in pixels.
- Select a value here only if you are printing to a window (if Destination = 0).

Availability
Design time; Runtime

WindowLeft

Description
Sets the distance, in pixels, that the print window is to appear from the left edge of the parent window. If the print window is a top level window, then the distance is measured from the left edge of the screen.

Runtime Usage
[form.]CrystalReport.WindowLeft[= Distance%]

For example:

```
CrystalReport1.WindowLeft = 100
```

«sets the left edge of the print window 100 pixels from the left edge of the screen.»

Remarks

- If you are not satisfied with the default settings, enter the number of pixels you want between the left edge of the screen and the left edge of your window.
- Select a value here only if you are printing to a window (if Destination = 0).

Availability
Design time; Runtime

WindowMaxButton

Description
Specifies whether or not the print window is to have a maximize button when the report is printed to a window.

Runtime Usage
[form.]CrystalReport.WindowMaxButton[= {True|False}]

For example:

```
CrystalReport1.WindowMaxButton = False
```

«specifies that no Maximize button is to appear in the print window.»

Remarks

- Select True if you want the window to contain a maximize button. Select False if you don't.
- Select a value here only if you are printing to a window (if Destination = 0).

Availability
Design time; Runtime

WindowMinButton

Description
Specifies whether or not the print window is to have a minimize button when the report is printed to a window.

Runtime Usage
[form.]CrystalReport.WindowMinButton[= {True|False}]

For example:

```
CrystalReport1.WindowMinButton = True
```

«specifies that a Minimize button is to appear in the print window.»

Remarks

- Select True if you want the window to contain a minimize button. Select False if you don't.
- Select a value here only if you are printing to a window (if Destination = 0).

Availability
Design time; Runtime

WindowParentHandle

Description
Specifies the handle of the parent window if the print window is to be the child of another window.

Runtime Usage
[form.]CrystalReport.WindowParentHandle[= ParentHandle%]

Remarks
This is a runtime-only property.

For example:

```
CrystalReport1.WindowParentHandle = Form1.hWnd
```

«sets the WindowParentHandle to the handle of Form1. This specifies that the print window is to be a child of Form1»

Availability
Runtime

WindowState

Description
Sets the state of the Preview window, normal, minimized, or maximized, when the report is printed to a Preview window.

Runtime Usage
[form.]CrystalReport.WindowState[= State%]

For example:

```
CrystalReport1.WindowState= 2
```

«When the report is printed to a Preview window, the Preview window appears maximized when opened.»

Remarks

Use the following values to set the WindowState property:

0 = Normal	The Preview window appears neither minimized nor maximized. It appears in a default size and position previously defined by your application or by Windows.	crptNormal
1 = Minimized	The Preview window appears minimized as an icon close to the lower left hand corner of the screen. The icon can be restored to display the window in a normal state.	crptMinimized
2 = Maximized	The Preview window is maximized when opened to fill the entire screen.	crptMaximized

Availability

Design time; Runtime

WindowTitle

Description

Specifies the title you want to appear in the print window title bar when the report is printed to a window.

Runtime Usage

[form.]CrystalReport.WindowTitle[= Title$]

For example:

```
CrystalReport1.WindowTitle = "Quarterly Earnings"
```

«sets the title of the print window (the string that appears on the title bar) to "Quarterly Earnings"»

Remarks

- Make sure that the title is enclosed in quotes.
- Select a value here only if you are printing to a window (if Destination = 0).

Availability

Design time; Runtime

WindowTop

Description

Sets the distance, in pixels, that the print window is to appear from the top edge of the parent window. If the print window is a top level window, then the distance is measured from the top edge of the screen.

Runtime Usage

[form.]CrystalReport.WindowTop[= Distance%]

For example:

```
CrystalReport1.WindowTop = 100
```

«sets the top edge of the print window 100 pixels from the top of the screen.»

Remarks

- If you are not satisfied with the default setting, enter the number of pixels you want between the top of the screen and the top of your window.
- Select a value here only if you are printing to a window (if Destination = 0).

Availability

Design time; Runtime

WindowWidth

Description

Specifies the width of the print window in pixels.

Runtime Usage

[form.]CrystalReport.WindowWidth[= Width%]

For example:

```
CrystalReport1.WindowWidth = 480
```

«specifies a print window 480 pixels wide.»

Remarks

- If you are not satisfied with the default setting, enter the external width of your window, in pixels.
- Select a value here only if you are printing to a window (if Destination = 0).

Availability

Design time; Runtime

Crystal ActiveX Control Methods

PrintReport

Description

PrintReport triggers the printing of the report.

Runtime Usage

[form.]CrystalReport.PrintReport

For example:

```
Result% = CrystalReport1.PrintReport
```

«prints the specified report.»

Remarks

- PrintReport returns a result code, 0 if the call is successful, an error code in the 20XXX range if it fails.
- You can also print a report using the Action property. If something goes wrong, however, you get a runtime error that will terminate your application. For this reason, you will need to set up an error handler.

Availability

Runtime

APPENDIX A

Product Support, Upgrades, and Licensing

Product Support

Crystal Reports is a very powerful report writer for the Windows environment. Seagate Software IMG, Inc. is proud of the quality of the product and has spent a great deal of time trying to make it intuitive to use. If, however, there is something you cannot figure out how to do, we suggest you consult the extensive Crystal Reports online Help System. The Help System can be accessed by clicking on any Help button, by pressing the F1 key, or by choosing the Contents command from the main Crystal Reports Help menu. The help system contains all of the information from the User's Guide, as well as in-depth explanations and numerous examples. If you still can not find an answer to your question in the User's Guide, we suggest you consult your Crystal Reports online Help. The help system includes an extensive index to help you find the topic of interest.

NOTE: Product support plans vary from region to region. Contact your local distributor for a list of product support plans available in your region. See README.HLP for a list of International distributors.

Facts On Demand

If you have a fax machine, you can get technical and marketing documents from Seagate Software IMG, Inc. by calling (604) 681-3450. Listen to the menu, select the documents you would like to receive, and enter your fax number. Seagate Software IMG, Inc. will fax the documents to you automatically.

NOTE: Product support plans vary from region to region. Contact your local distributor for a list of product support plans available in your region. See README.HLP for a list of International distributors.

Seagate Facts On Demand (604) 681-3450

Fax Support - FREE

Another efficient way to receive support on Crystal Reports is to fax in your technical support request. To do this, first fill in the bottom part of the Technical Support Request form in the product. This form appears when you choose the Technical Support Request command from the Crystal Reports Help menu. When you have completed the form, click the Print button to print it.

Fax the completed form to us 24 hours a day, Monday through Saturday. After a technical support representative has had a chance to review your fax, one of our support representatives will respond to you by return fax.

NOTE: Product support plans vary from region to region. Contact your local distributor for a list of product support plans available in your region. See README.HLP for a list of International distributors.

Seagate Fax Support (604) 681-7163

Mail-in Support - FREE

If you prefer, you can contact Seagate Software IMG by mail with written questions or comments, and we will respond by return mail.

If you believe that your questions are such that we should review all of your related files (such as databases, etc.) to help solve your problem, then copy all of the related files and the Crystal Reports report file (FILENAME.RPT) onto a diskette of any PC compatible format and send it to us. We will investigate the problem and mail back a response to you as soon as possible with your original diskette.

NOTE: Product support plans vary from region to region. Contact your local distributor for a list of product support plans available in your region. See README.HLP for a list of International distributors.

Our mailing address is:

Seagate Software IMG, Inc.
4th Floor - 1095 West Pender Street
Vancouver, BC, Canada V6E 2M6

Telephone Support

Registered users of Crystal Reports are entitled to free telephone support (subject to availability) for 60 days from the time of purchase. Telephone support is available from 8 a.m. to 5 p.m., P.S.T. Monday through Thursday and 8 a.m. to 4 p.m. P.S.T. on Friday. If you are calling from out of the Vancouver, BC area, long distance charges apply.

Before you call technical support, make sure you do the following:

- check the manual
- check help
- check FOD (Facts On Demand)
- check the Internet at http:\\www.img.seagatesoftware.com\

PLEASE HAVE THE FOLLOWING INFORMATION AVAILABLE:

- serial number,
  - If you are registered, you can find the serial number by choosing the About Crystal Reports command on the Help menu.
  - If you are not registered, you will need to register first to obtain your serial number.
- product name and version number,
- operating system you are using (i.e., Windows 95 or Windows NT),
- version of database and other software you are using with Crystal Reports (if required) - Btrieve, Sybase, Paradox, etc.,
- Technical Support Request form from the product,
- network information if you are on a network,
- contents of autoexec.bat and config.sys files,
- the of steps necessary to recreate the problem,
- the database type you are using, and
- the programming environment (if applicable).

NOTE: Product support plans vary from region to region. Contact your local distributor for a list of product support plans available in your region. See README.HLP for a list of International distributors.

Telephone Number (604) 669-8379

Extended Technical Support Policy

Registered users may purchase an extended support policy which entitles the user to unlimited technical support for a one year period. Support plan members will receive a special phone number which they can use to get directly through to the Technical Support team. Call (604)681-3435 for detailed information about this plan.

NOTE: The extended technical support policy is available in the United States and Canada only.

Product Registration

When you use Crystal Reports for the first time, it will ask for your name, address, and related information. Then the product will suggest that you register Crystal Reports to receive your product Serial Number. The program will ask for the Serial Number the next time you run Crystal Reports. When you enter the Serial Number, you have completed the registration process. You can register Crystal Reports using three different methods:

1. Fill out the Registration form that is built into Crystal Reports and then register by modem using the Crystal Reports communications program. The procedure will register your copy of the program, assign a registration number, and enter that number automatically into your system.
2. Print the Registration Form that is built into Crystal Reports and then fax it to us at (604) 681-5147. We will then fax back to you a registration number that can be entered into the product the next time it asks for it.
3. Fill out the enclosed Registration Form and mail it to us. We will mail a registration number back to you so you can enter it into the product.

Registering Crystal Reports will ensure that you are kept up to date with all product advancements, and it will allow us to provide quality technical support to people that are properly registered with Crystal Services.

Registration FAX (604) 681-5147

Crystal Reports Upgrade Plan

If you would like to automatically receive upgrades to Crystal Reports, you can purchase Crystal Reports Upgrade plan. This will entitle you to receive free upgrades for one year for only $150.00 per year. You will automatically get upgrades shipped by UPS ground. (You can receive them by air, if you wish, at an additional expense.) This upgrade plan includes all product upgrades as well as any documentation changes during the year. If you want worry free software maintenance, then this is the plan for you.

NOTE: The upgrade plan is available in the United States and Canada only.

Order Phone Line (604) 681-3435
Order FAX line (604) 681-2934

Product Replacement Policy

If the Crystal Reports diskettes or documentation are defective, then please contact Seagate Software Information Management Group, Inc. within 30 days of the purchase date. Fax the description of the problem and we will solve it as quickly as possible. Please fax the description of the defect to (604) 681-2934.

NOTE: Replacement policies vary from region to region. Contact your local distributor for a list of replacement policies available in your region. See README.HLP for a list of International distributors.

License Agreement And Limited Warranty

IMPORTANT - READ CAREFULLY

This License Agreement and Limited Warranty (herein referred to as "Agreement") is a legal agreement between you (either an individual or a single entity) and Seagate Software Information Management Group, Inc. (herein referred to as "Seagate Software") for the Crystal Reports software product, which includes computer software and associated media, printed materials, and "on-line" or electronic documentation (herein referred to as "Software Product"). By opening the sealed software package and/or by installing, copying or otherwise using the Software Product, you agree to be bound by the terms of this Agreement. If you do not agree to the terms of this Agreement, promptly return the Software Product and accompanying items in resalable condition to the place from which you obtained them for a full refund.

1. LIMITED LICENSE.

   (a). You are purchasing only the right to use a single copy of the Software Product, and are acquiring no rights to the software itself, or the enclosed copy, except the Limited License granted herein.

   (b). Except as stated in (c), Seagate Software grants you a Limited License to use one copy of the Software Product on a single computer. You may access the Software Product from a hard disk, over a network (subject to (c)) or by any other method, strictly in accordance to the terms hereof.

   (c). For additional users to use the Software Product on a network, you must acquire "Multiple User Packs", which allow additional designated users in numbers and on terms specified by the documentation accompanying the Software Product. You may have as many copies of the Software Product in use on a network as is specifically authorized in the documentation.

2. COPYRIGHT. All title and copyrights in and to the Software Product (including but not limited to, any images, photographs, animations, video, audio, music, and text incorporated into the Software Product), the accompanying printed materials, and any copies of the Software Product are owned by Seagate Software or its suppliers. The Software Product is protected by copyright laws and international treaty provisions. Therefore, you may not copy the Software Product except that you may either (a) make one copy of the Software Product solely for backup or archival purposes, or (b) install the Software Product on a single computer provided you keep the original solely for backup or archival purposes. You may not copy the printed materials accompanying the Software Product.

3. OTHER RESTRICTIONS.

(a). Rental. You may not rent or lease the Software Product.

(b). Software Transfer. Unless this is a "Not for Resale" or "Evaluation" product, you may permanently transfer all of your rights under this Agreement, provided that you retain no copies, you transfer all of the Software Product (including all component parts, the media and printed materials, any upgrades, and this Agreement), and the recipient agrees to the terms of this Agreement. If the Software Product is an upgrade, any transfer must include all prior versions of the Software Product.

(c). Export. Any export of the Software Product or its technical data from the United States may be restricted by U.S. export control laws, and accordingly you must comply with such laws before engaging in any such export.

(d). Limitations on Modifications or Reverse Engineering. You may not alter, disassemble, decompile, or reverse-engineer the Software Product.

(e). No Separation of Components. The Software Product is licensed as a single product, and neither the software programs comprising the Software Product or any update or upgrade may be separated for use by more than one user at a time.

(f). Termination. Without prejudice to any other rights, Seagate Software may terminate this Agreement if you fail to comply with the terms and conditions of this Agreement. In such event, you must destroy all copies of the Software Product.

4. LIMITED WARRANTY.

(a). Seagate Software warrants the physical disk(s) and physical documentation to be free of defects in materials and workmanship for a period of 60 days from the purchase date. If Seagate Software receives notification within this warranty period of any such defects and such notification is determined to be correct, Seagate Software will replace the disk(s) or documentation.

(b). Seagate Software's entire liability for any breach of this warranty is limited to replacement of defective disk(s) or documentation, and does not include or extend to claims for any other damages.

(c). SEAGATE SOFTWARE SPECIFICALLY DISCLAIMS ALL OTHER CONDITIONS AND WARRANTIES, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED CONDITION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

5. LIMITATION OF LIABILITY. IN NO EVENT SHALL SEAGATE SOFTWARE BE LIABLE FOR COSTS OF SUBSTITUTE PRODUCTS, LOSS OF PROFITS, DATA OR USE OF THE SOFTWARE PRODUCT, OR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OR OTHER SIMILAR CLAIMS, HOWEVER CAUSED, AND ON ANY THEORY OF LIABILITY, EVEN IF SEAGATE SOFTWARE HAS BEEN SPECIFICALLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT WILL SEAGATE SOFTWARE'S LIABILITY FOR ANY DAMAGES TO YOU OR ANY OTHER PERSON EVER EXCEED THE LOWER OF THE SUGGESTED LIST PRICE OR THE ACTUAL PURCHASE PRICE PAID FOR THE SOFTWARE PRODUCT, REGARDLESS OF THE FORM OF THE CLAIM. THESE LIMITATIONS SHALL APPLY NOTWITHSTANDING ANY FAILURE OF ESSENTIAL PURPOSE OF ANY LIMITED REMEDY. ALL THIRD PARTY SUPPLIERS TO SEAGATE SOFTWARE OF ANY PORTION OF THE SOFTWARE PRODUCT MAKE NO WARRANTY AND SHALL HAVE NO LIABILITY TO YOU.

6. RUNTIME SOFTWARE. The software files listed in the software file RUNTIME.HLP, provided on your software disk(s), individually, collectively or in any combination, constitute Runtime Software. Unless this is a "Not for Resale" or "Evaluation" product, Seagate Software grants you a Limited Runtime License to reproduce and distribute exact copies of the Runtime Software, and to modify, reproduce and distribute the sample applications made from the original disks in this package (or a set of backup copies made directly from them), if and only if all of the following conditions are satisfied:

   (a). you may distribute such copies solely for the purpose of executing specific-purpose application programs written using an authorized copy of Software Product;

   (b). you remain solely responsible for support, service, upgrades, and technical or other assistance, required or requested by anyone receiving such copies;

   (c). you may not use the name, logo, or trademark of Seagate Software or the Software Product without written permission from Seagate Software;

   (d). you may not alter, disassemble, decompile, or reverse-engineer the Runtime Software;

   (e). Seagate Software makes no warranty other than the Limited License warranty provided in this agreement to you; and

   (f). you will indemnify Seagate Software against any claims or liabilities arising out of its use, reproduction or distribution, and the use by the recipients or copies made thereunder.

You may not include the Runtime Software or sample applications in any general-purpose report writing and/or database querying product, or any other product that is generally competitive with Seagate Software product offerings. If you wish to distribute copies of the Runtime Software or sample applications on other terms, you must obtain written permission from Seagate Software.

"Not for Resale" and "Evaluation" products do not include a Limited Runtime License, and you may not, therefore, distribute Runtime Software or sample applications.

The Software Product contains software components from Seagate Software suppliers. This Agreement does not grant you the rights to distribute software components from these suppliers, including but not limited to, Microsoft Corporation's "JET Files", unless you have proper authorization from the appropriate supplier.

7. GOVERNING LAW. If you acquired this product in Canada, this Agreement is governed by the laws of the Province of British Columbia, Canada, without regard, and each party irrevocably attorns to the exclusive jurisdiction of the courts of the Province or British Columbia. If you acquired this product in the United States or elsewhere other than Canada, this Agreement is governed by the laws of the State of California, without regard, and each party irrevocably submits to the exclusive jurisdiction of the courts of the State of California.
8. MISCELLANEOUS. This document constitutes the entire agreement of the parties regarding the subject matter herein. All rights not expressly granted to you are retained by Seagate Software.

Restricted Rights Legend

U.S. GOVERNMENT RESTRICTED RIGHTS. Use, duplication, reproduction, or transfer of this commercial product and accompanying documentation is restricted in accordance with FAR 12.212 and DFARS 227.7202 and by this license agreement. Contact Legal Department, Seagate Software, Inc. (408) 456-4646.

International Partners Directory

Argentina

SOFTLAND
Mansilla 2437 1 er piso
(1121) Buenos Aires
Argentina
Ph: 54-1-964-2100
Fax: 54-1-964-2270
E-mail: Info@softland.com.ar
Web: www.softland.com.ar
Intranet: 964-2280
AutoFax: 964-2290 (Fax on Demand System)

Australia

AERONAUT INDUSTRIES
Ground Floor
48 Chandos Street
St Leonards NSW2065
Ph: 61-2-9436-1175
Fax: 61-2-9436-1184
E-mail: sales@aeronaut.com.au

Austria

Contact Seagate Software Germany

MICRO BASIC
Am Sommerfeld 11
D-85622 Weissenfeld
Ph: 49-89-9049-9049
Fax: 49-89-9049-9090

Belgium

Contact Seagate Software France.

Brazil

RAMO SISTEMAS DIGITAIS
Rua Haddock Lobo 337 8 andar
01414-001 Sao Paulo, SP Brazil
Ph: 55-11-214-2278
Fax: 55-11-214-2565
E-mail: ramosist@embratel.net.br

Chile

BRIGHT SYSTEMS
Dr. Manuel Barros Borgono 163
Providencia, Santiago
Ph: 56-2-235-0480
Fax: 56-2-236-1069
E-mail: bsystem@entelchile.net

Columbia

ESTRATEGIAS Y SISTEMAS CIA LTDA.
Carrera 37 No. 89-99
A.A. 251312 Santafe de Bogota Colombia
Ph: 57-1-236-1696
Fax: 57-1-617-7530
E-mail: eysjosem@openway.com.co

Cyprus

CYCOM COMPUTER PRODUCTS LTD.
City House, 19th Dervis St.
P.O. Box 5732
Nicosia, Cyprus
Ph: 357-2-454448
Fax: 357-2-454707

Czech Republic

SWS SOFTWARE s.r.o.
Dostihova 1
763 15 Slusovice
Ph: 42-67-7981036
Fax: 42-67-7981008

ABAKUS DISTRIBUTION A.S.
Maltezske nam. 7
110 00 Praha 1
Ph: 0042-2-245 10 942
Fax: 0042-2-245 10 254

Denmark

NORSOFT A/S
Maren Turis Gade 2
DK-9100, Aalborg
Ph: 45-98-11-1188
Fax: 45-98-11-1088

Finland

BOSS INFORMATION
Sepankatu 14 A
SF-40720, Jyvaskyla
Ph: 358-41-334-3411
Fax: 358-41-334-3444

France

SEAGATE SOFTWARE S.A.
62bis Avenue André Morizet
92643 Boulogne-Billancourt Cedex
Ph: 33-1-41-10 16 00
Fax: 33-1-46-04 74 19

Distributor:

SOFTWARE TECHNOLOGY RESOURCES
7 rue Paul Dautier
78140, Velizy
Ph: 33-1-3070-6161
Fax: 33-1-3946-3864

Germany

SEAGATE SOFTWARE GMBH
Jacob-Kaiser-Str. 7
47877 Willich
Ph: 49-0-2154-916350
Fax: 49-0-2154-916370

Distributors:

MICRO BASIC
Am Sommerfeld 11
D-85622 Weissenfeld
Ph: 49-89-9049-9049
Fax: 49-89-9049-9090

MICROWARE GMBH
Moselstrasse 21
41464 Neuss
Ph: 49-2131 94 45-0
Fax : 49-2131 4 37 04

Greece

Contact Seagate Software France.

Orthology Consulting Ltd.
8 Heroon Polytechneiou
Aghia Paraskevi 153 42
ATHENS, GREECE
Ph: +301 60.80.091-2
Fax: +301 60.80.092
e-mail: orthology@ath.forthnet.gr
Web: http://www.forthnet.gr/orthology

Hong Kong

DATADEVELOP LTD
Rm 1001-3 Fourseas Bldg.
208-212 Nathan Road
Kowloon, HongKong
Ph: 852-2-312-6038
Fax: 852-2-314-3672
E-mail: dglpsd@nkstar.com

Hungary

Questar Hungary
Huvosvolgyi ut 54
H-1021 Budapest
Ph: 36-1-202-6373
Fax: 36-1-202-6375

India

LARSEN & TOUBRO LTD.
Shailesh Mahajan
Powal, Bombay 400 072
Ph: 91-22-852-3309
Fax: 91-22-852-1592
E-mail: akc-isd@smtpg.Itindia.com

Israel

Bynet
8 Hanechoshet St.
Ramot - Hachayal
Tel Aviv, Israel
Ph: 972-3-645-8080
Fax: 972-3-648-1656

Italy

Contact Seagate Software France.

QUESTAR
Via Ghislandi, 61/B
24125 Bergamo
Ph: 39-35-24-9946
Fax: 39-35-24-9945
E-mail: info@questar.it
Web: www.questar.it

Japan

AG TECH
Bell Ace Building 4F
1-10-9 Higashi Sakura Higashi-Ku
Nagoya 461
Ph: 81-52-951-2706
Fax: 81-52-951-4469
E-mail: 72041.1166@compuserve.com

WOODLAND CORP.
Asahiseimei-imabashi bldg. 1-6-19
Imabashi, chuo-ku
Osaka 541
Ph: 81-6-204-0611
Fax: 81-6-229-0506
E-mail: shimada@woodland.co.jp

Korea

INFOTECH
5 Floor Wha Building
238, Nonhyon-Dong, Kangnam-ku
Seoul 135-010
Ph: 82-23-442-0400
Fax: 82-23-442-1960

CC COM
Rodem Bldg 3F
834-77 Yeosam-dong, Kangnam-gu
Seoul 135-080
Ph: 82-2-538-8700
Fax: 82-2-538-8716
E-mail: 100053.377@compuserve.com

Luxembourg

Contact Seagate Software France.

Mexico

OPENTEC SA de CV
Insurgentes Sur 2047-4
Col.San Angel
01000 Mexico D.F. Mexico
Ph: 52-5-661-4472
Fax: 52-5-661-3571
E-mail: CMEJIA@opentec.com.mx

New Zealand

UNILOGIC
195 Khyber Pass Rd, Level 4
Grafton, Auckland
Ph: 64-9-358-8114
Fax: 64-9-358-1224
E-mail: crystal@unilogic.co.nz
Web: www.unilogic.co.nz

Netherlands

Contact Seagate Software UK.

Norway

COMPUTER RESOURCES INTERNATIONAL
Postboks 139 Kjelsas
0411 Oslo
Ph: 47-22-236893
Fax: 47-22-952355

POLYSOFT A/S
Rosenkrantsgt. 95
Box 829
N-3007 Drammen
Ph: 47-3289-2240
Fax: 47-3289-5900

Philippines

PACIFIC COMPUTER & MANAGEMENT SYSTEMS
5E Vernida 1 Bldg, 120 Amorsolo St. Legaspi Village, Makati City, MM
Ph: 632-892-2148
Fax: 632-818-3807
E-mail: titodel@helios.iconn.com.ph

Poland

POLCOM SP Z.O.O.
ul. G. Zapolskiej 38
30-126 Krakow
Poland
Ph: 0048-12-36 71 77
Fax: 0048-12-37 60 62

Portugal

PRIMAVERA SOFTWARE
R. Cidade Do Porto no.15, Sala E-11
4700 Braga
Ph: 351-53693169
Fax: 351-53693334

Russia

Contact Seagate Software Germany.

Singapore

NOVA SPRINT CONSULTING
Block 1002 Jalan Bukit Merah, #03-16/20 Redhill Industrial Estate,
159456
Ph: 65-275-9088
Fax: 65-275-9388
E-mail: laitk@mcsb.com.sg

South Africa

SEAGATE SOFTWARE
376 Oak Avenue
Randburg 2125
Ph: 27-11-781-2200
Fax: 27-11-781-2214

Spain

Contact SeagateSoftware France.

RAMBLA INFORMATICA
Primer de Maig 2-22 baixos
08320, El Masnou, Barcelona
Ph: 34-3-540-2982
Fax: 34-3-540-3095

Danysoft Internacional
Pase De Albacete, 73
28700 S.S. De Los Reyes
Madrid, Spain
Ph: +341-654-6298
Fax: +341-654-6382

Sweden

LINSOFT
S:t Largatan 9 D
Linkoping
Ph: 461-311-1588
Fax: 461-313-5695

Switzerland

Contact Seagate Software Germany

MICRO BASIC
Am Sommerfeld 11
D-85622 Weissenfeld
Ph: 49-89-9049-9049
Fax: 49-89-9049-9090

Taiwan ROC

WARDPRO INFORMATION CORP.
4F-1,No 20 Alley 1 Lane 768
Section 4 Pa-Teh Road, Taipei, Taiwan
Ph: 886-2-788-6777
Fax: 886-2-785-3220

Thailand

INDEC TECHNOLOGY COMPANY
436 Soi 12
Paholyotin Road
Bangkok 10400
Phone: 662-513-1077
Fax: 662-279-1556

SAHASEDSIRIRI LTD. PARTNERSHIP
2104/48 Soi Mooban Decha Ramkamaeng Rd.
Huamark Bangkapi, Bangkok 10240
Ph: 662-732-3268
Fax: 662-374-7163
E-mail: narun@ksc6.th.com

Turkey

OPTIMUM
Barboros Bulvari Akdogan Sok. 37/3
80690 Besiktas, Istanbul
Ph: 90-212-2581829
Fax: 90-212-2583567

United Arab Emirates

MITECH TRADING
P.O. Box 4886, Dubai
Ph: 971-4-624043
Fax: 971-4-665818

United Kingdom

SEAGATE SOFTWARE UK
Kings Chase, 107 King Street
Maidenhead, Berkshire
SL6 1DP
Ph: 44-1628-771299
Fax: 44-1628-771523

Distributors:

CONTEMPORARY SOFTWARE
Kingswick House, Sunninghill, Berkshire,
SL5 7BH
Ph: 44-1344-873434
Fax: 44-1344-872228
E-mail: cssales@contemporary.co.uk
Web: www.contemporary.co.uk

IDEAL HARDWARE
265 Burlington Road
New Malden, Surrey
KT3 4NE
Ph: 44-181-286 5000
Fax: 44-181-286 5056
Web: www.ideal.co.uk

Venezuela

L & C COMPUTER SERVICES, C.A
Calle El Porvenir, Edificio Tocome, piso 1, oficina 2A,
Urb. La Florida, Caracas 1050 Venezuela
Ph: 58-2-731-2668
Fax: 58-2-731-2668
E-mail: arturolaos@true.net

Microsoft®